Authentication and Encryption Protocols: Design, Attacks and Algorithmic Improvements

Thèse de Doctorat

en vue de l'obtention du grade de

Docteur de l'École normale supérieure
(spécialité informatique)

présentée et soutenue publiquement le 11 Decembre 2015 par

DIANA ŞTEFANIA MAIMUŢ

devant le jury composé de :

Directeur de thèse :	David Naccache	(École normale supérieure)
Rapporteurs :	Bart Preneel	(iMinds, Katholieke Universiteit Leuven)
	Moti Yung	(Google Inc., Columbia University)
Examinateurs :	Jean-Sébastien Coron	(Université du Luxembourg)
	Antoine Joux	(UPMC, Université Paris VI)
	David Pointcheval	(CNRS, École normale supérieure)
	Serge Vaudenay	(École polytechnique fédérale de Lausanne)

École doctorale 386: Sciences mathématiques de Paris Centre
Unité de recherche: UMR 8548 - Département d'Informatique de l'École normale supérieure
Laboratoire de recherche affilié au CNRS et a INRIA

Authentication and Encryption Protocols: Design, Attacks and Algorithmic Improvements

Thèse de Doctorat

en vue de l'obtention du grade de

Docteur de l'École normale supérieure
(spécialité informatique)

présentée et soutenue publiquement le 11 Decembre 2015 par

DIANA ŞTEFANIA MAIMUŢ

devant le jury composé de :

Directeur de thèse :	David Naccache	(École normale supérieure)
Rapporteurs :	Bart Preneel	(iMinds, Katholieke Universiteit Leuven)
	Moti Yung	(Google Inc., Columbia University)
Examinateurs :	Jean-Sébastien Coron	(Université du Luxembourg)
	Antoine Joux	(UPMC, Université Paris VI)
	David Pointcheval	(CNRS, École normale supérieure)
	Serge Vaudenay	(École polytechnique fédérale de Lausanne)

École doctorale 386: Sciences mathématiques de Paris Centre
Unité de recherche: UMR 8548 - Département d'Informatique de l'École normale supérieure
Laboratoire de recherche affilié au CNRS et a INRIA

Acknowledgements

My very first thoughts are dedicated to my parents who have encouraged me in every step I took in research. I express my infinite gratitude towards my father who left us earlier than expected and who would have wished to see this thesis finished. Thanks to my mother especially for teaching me not to give up easily.

I would like to thank *David Naccache* for his guidance as my thesis advisor during these four years. You opened many doors for me, showing endless patience and support. I am also grateful to *David Naccache* for allowing me to formally supervise master theses in his Information Systems Forensics Master (*Expertise économique et juridique des systèmes d'information*).

I also thank my co-authors, *Simon Cogliani, Eric Brier, Jean-Sébastien Coron, Houda Ferradi, Cédric Murdica, Khaled Ouafi, David Pointcheval, Reza Reyhanitabar, Mehdi Tibouchi, Serge Vaudenay, Damian Vizár, Amaury de Wargny* and *Hang Zhou*. Special thanks to *Zineb Benkirane*, *Marc Beunardeau*, *Rémi Géraud* and *Rodrigo Portella do Canto*, it was truly a pleasure working with you.

I would like to express my appreciation towards the members of the crypto team, be it professors or *jeunes chercheurs*. Thank you for your welcoming and stimulating team spirit, for the scientific creativity, the truly international nature of the team and the constant pushing of intellectual boundaries. I am especially indebted to *David Pointcheval* for his support throughout the entire course of this thesis.

I have to thank *Emil Simion* for his useful comments and advices during the last years and, moreover, for encouraging me to apply to ENS for my PhD studies. I am indebted to Emil Simion for actively involving me in SECITC 2015. The review of the submissions and the intense technical interaction with the program committee members were an unforgettable scientific experience. I am also grateful to *Adrian Atanasiu, Dorin Popescu, Cristian Voica* and *Victor Vuletescu* for guiding me in my early academic life and for showing me the real beauty in mathematics. Thanks to *Alecsandru Pătraşcu* for his valuable comments on some of my papers. You truly are a promising young researcher.

I am mostly grateful to *Jean-Sébastien Coron, Antoine Joux, David Pointcheval* and *Serge Vaudenay* for agreeing to serve in this thesis committee. I express my particular gratitude to my thesis referees *Bart Preneel* and *Moti Yung* for their availability and dedication. I am very honoured to have such a prestigious committee.

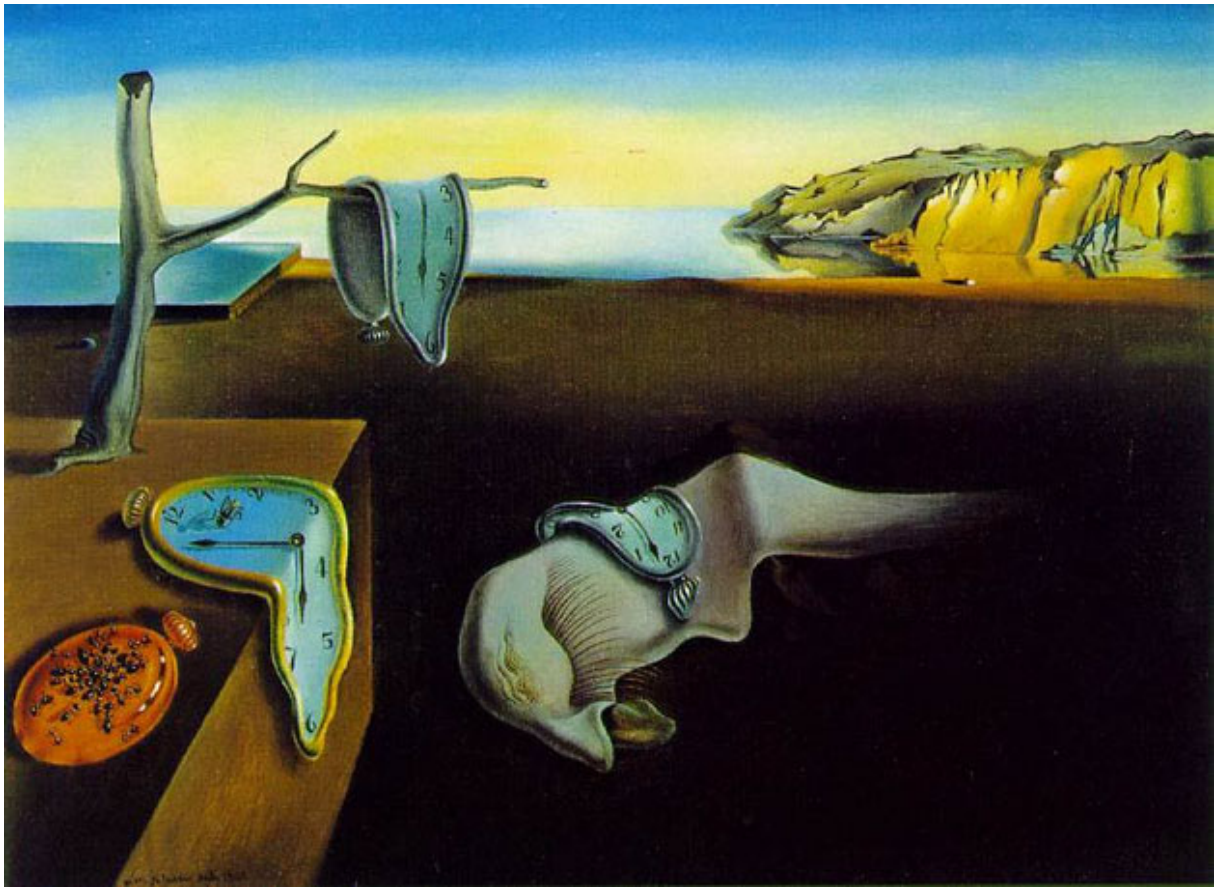
The research works presented in this thesis were supported by two high-tech firms. Helping R&D to apply my research results in commercial products was a thrilling experience, for which I warmly thank *Ingenico* and *Crocus Technology*.

I express my gratitude towards my current employer, the *Advanced Technologies Institute*. Thank you for supporting me especially towards the end of this thesis, and for offering me the possibility to apply the knowledge I accumulated.

Last but most certainly not least, I would like to thank *Natacha Laniado* for being such a wonderful person. You supported me since the very beginning, helped me whenever I needed, proofread my papers and thesis. Wishing you all the best.

How long do you want these messages to remain secret?[...]
I want them to remain secret for as long as men are capable of evil.

– Neal Stephenson



The Persistence of Memory – Salvador Dali.

Contents

CHAPITRE 1

PROLÉGOMÈNES

If you reveal your secrets to the wind, you should not blame the wind for revealing them to the trees.
Kahlil Gibran.

Résumé

Ce chapitre présente la terminologie nécessaire à la lecture de cette thèse et positionne la cryptographie dans le domaine plus large de la cryptologie. Nous détaillerons les plus importantes étapes de l'histoire de la cryptologie, partant de l'antiquité et arrivant à la cryptographie post-quantique avec une insistance particulière sur les notions de cryptographie symétrique et asymétrique. La section 1.2 décrit de manière synthétique les sujets abordées dans cette thèse et fixe les objectifs à atteindre. Nos principaux résultats scientifiques sont des nouveaux protocoles de signature et d'authentification et un nouvel algorithme de chiffrement authentifié. Nos travaux comportent également des résultats liés aux codes correcteurs d'erreurs, à la correction de biais lors de la génération d'aléa, à la cryptanalyse et plusieurs améliorations calculatoires.

Notre premier résultat est un protocole de co-signature prouvé sûr, réalisant une nouvelle forme d'équité dite *équité légale*. Le second est un algorithme de chiffrement authentifié prouvé sûr nommé Offset Merkle-Damgård (OMD). OMD a la rare particularité d'être un mode d'opération pour une fonction de compression à clé. OMD est actuellement l'un des finalistes de la seconde phase de sélection de la compétition internationale CAESAR.

En outre, nous présentons un protocole distribué de type Fiat-Shamir permettant l'authentification sur des réseaux. Nos améliorations algorithmiques comportent plusieurs résultats. Nous décrirons un nouvel algorithme de multiplication basé sur le retour sur trace. Cet algorithme s'avère particulièrement adapté aux microprocesseurs bon marché. Une seconde observation permet de doubler la vitesse de l'algorithme de Barrett à l'aide de modules composites spécifiques et une troisième contribution permet de régulariser le débit d'extracteurs de von Neumann. Nous présenterons également des nouvelles stratégies d'accélération de codes BCH utilisant des versions polynomiales de l'algorithme de Barrett ainsi qu'un nouveau code correcteur d'erreurs inspiré par le système de chiffrement Naccache-Stern. Enfin, nous décrirons une nouvelle attaque en fautes sur les algorithmes de signature à courbes elliptiques.

1.1 Une brève introduction à l'histoire de la cryptographie

Selon le dictionnaire de Merriam-Webster [MWb], le terme *cryptologie* a été utilisé pour la première fois en 1935 et se réfère à *l'étude scientifique de la cryptographie et de la cryptanalyse*. Le mot *cryptographie* est apparu en 1658 et son origine est étroitement liée au concept du latin moderne cryptographia qui a hérité du grec les termes *kryptos* (caché, secret) et *graphein* (écrire). La première utilisation du mot *cryptanalyse* date de 1923 [MWa].

La cryptographie vise à créer des systèmes préservant des informations secrètes ou inaltérées. La cryptanalyse est l'art dual, consistant à mettre à défaut des systèmes cryptographiques.

Les objectifs traditionnels de la cryptographie sont:
– L'*intégrité*, garantissant l'inaltérabilité du message lors de sa transmission.
– L'*authentification*, s'assurant de l'identité de l'expéditeur du message.
– La *non-répudiation*, empêchant la répudiation des actions passées.

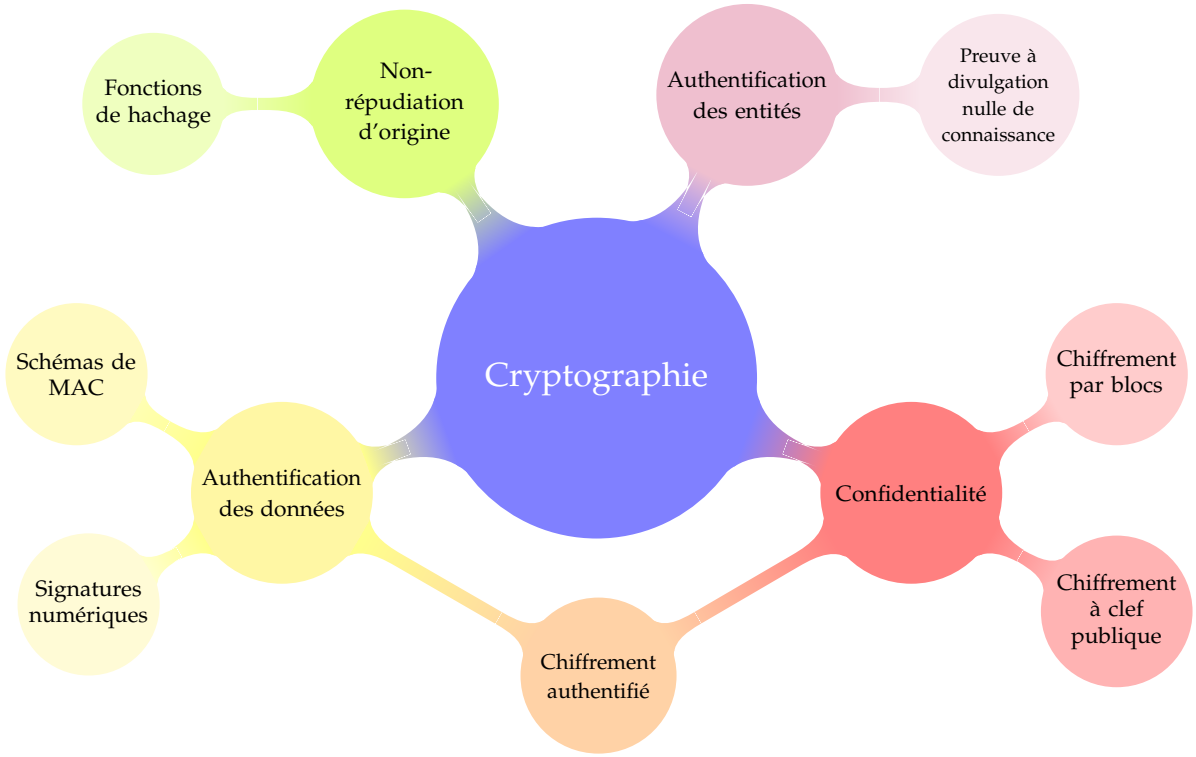


Figure 1.1: Quelques objectifs et primitives cryptographiques.

Le terme générique *systèmes cryptographiques* (ou *cryptosystèmes*) désigne l'ensemble des algorithmes accomplissant certains ou tous les objectifs mentionnés précédemment.

L'enjeu du secret est un invariant de la cité, de la société, de nos politiques. Il est à ce point focal que d'aucuns abordent le secret au titre de la raison dite d'État, ou encore au maintien d'un équilibre que sauvegardent, les diplomates, les militaires et les espions - qui sont les utilisateurs historique du chiffrement.

La récente éruption des télécommunications dans notre quotidien a rendu la cryptographie omniprésente.

Notations. Les notations suivantes seront utilisées dans l'ensemble du manuscrit:
– $\mathcal{M} = \{m | m \in \mathtt{Alph}_m^*\}$ désignera l'ensemble des textes clairs (aussi appelé *espace des messages*), où $\mathtt{Alph}_m$ est un alphabet non-vide.
– $\mathcal{C} = \{c | c \in \mathtt{Alph}_c^*\}$ désignera l'ensemble des textes chiffrés (aussi appelé *espace des textes chiffrés*), où $\mathtt{Alph}_c$ est un alphabet non-vide.
– $\mathcal{K}$ désignera l'ensemble des clés (aussi appelé l'*espace des clés*).
Habituellement, $\mathtt{Alph}_m = \mathtt{Alph}_c$.

1.1.1 Cryptographie pré-informatique

Les chiffrements classiques

Les codes secrets classiques sont historiquement importants. Cette catégorie d'algorithmes de chiffrement est communément désignée par l'expression "chiffrements avec papier et stylo" et comprend des algorithmes cryptographiques simples qui peuvent être mis en œuvre soit directement à la main soit au moyen d'outils mécaniques très simples.

L'un des exemples les plus souvent invoqués d'algorithmes de chiffrement anciens est la *Scytale*, originaire de Sparte en Grèce. La scytale lacédomienne (*skutalé*) est considéré comme l'ancêtre des systèmes de transmission d'information secrète. Le dispositif consistait en un bâton autour duquel était enroulée une bande de parchemin, sur laquelle le message était écrit sur toute la longueur. Populaires durant le Vème siècle avant JC, les *Scytales* appartiennent à la catégorie des chiffrements par transposition.

Le chiffre de César. Ce chiffrement éponyme de Jules César, date du Ier siècle avant JC. Le chiffre de César est un chiffrement par substitution monoalphabétique.

Considérons les 26 lettres de l'alphabet. Ainsi, $\texttt{Alph}_m = \texttt{Alph}_c = \mathbb{Z}_{26}$. Le chiffre de César décale cycliquement chaque lettre du texte claire de trois positions pour obtenir le chiffré. Le nombre de décalages peut être remplacé par toute clé de chiffrement $k \in \mathbb{Z}_{26}$. Le processus de déchiffrement est un décalage inverse.

Le chiffre de Vigenère. Le chiffre de Vigenère a été développé par Blaise de Vigenère au XVIème siècle [Kah96].

Considérons de nouveau les 26 lettres de l'alphabet et choisissons un clé κ de longueur r. Le chiffre de Vigenère peut être construit à partir de r chiffres de César différents (chacun correspondant à un caractère de κ). Si le message à chiffrer et plus long que κ, la clé est utilisée périodiquement.

Le chiffre de Vigenère est donc un système de substitution.

Le chiffre de VIGENÈRE a été développé par Blaise de Vigenère au 16ème siècle [Kah96].

Considérons de nouveau les 26 lettres de l'alphabet et choisissons un clé κ de longueur r. Le chiffre de VIGENÈRE peut être construit à partir de r chiffres de César différents (chacun correspondant à un caractère de κ). Si le message à chiffrer et plus long que κ, la clé est utilisée périodiquement.

Le chiffre de VIGENÈRE est donc un système de substitution.

Le masque jetable. Aussi connu sous le nom de chiffre de Vernam, peut être décrit comme un chiffre de Vigenère dont la longueur de la clé est la même que la longueur du texte clair. En outre, la clé doit être choisie aléatoirement et ne peut être utilisée qu'une seule fois.

Le masque jetable souligne le rôle important que les générateurs de nombres aléatoires jouent en cryptographie.

Des chiffrements par transposition. Les chiffrements par transposition sont encore un autre type de chiffrement. Clarifions avec un exemple de chiffrement par transposition en colonnes. Un tel chiffrement commence par diviser le message m en blocs de ℓ-bits l'écrivant par lignes dans un tableau de taille fixée. Après cela, il applique une permutation [1] σ au message. Le texte chiffré obtenu est lu par colonnes.

Dans la figure 1.2 un exemple est décrit avec $m = \texttt{EXPLAINTRANSPOSITIONCIPHERS}$, $\kappa = \texttt{SECRET}$ et $\texttt{X}$ est utilisé comme un caractère de padding. La permutation est donnée par l'ordre alphabétique sur les lettres de κ, ce qui résulte en $\sigma = (3, 2, 5, 4, 1, 6)$. Le texte chiffré est donc $\texttt{PRSCSXTONRANTPXLAIIXENPOEISIHX}$.

1. qui est fonction de la clé

S	E	C	R	E	T
E	X	P	L	A	I
N	T	R	A	N	S
P	O	S	I	T	I
O	N	C	I	P	H
E	R	S	X	X	X

C	E	E	R	S	T
P	X	A	L	E	I
R	T	N	A	N	S
S	O	T	I	P	I
C	N	P	I	O	H
S	R	X	X	E	X

Figure 1.2: Columnar transposition cipher example.

Chiffrements électriques et mécaniques

Pour automatiser le chiffrement et le déchiffrement, différents designs mécaniques ont été proposées, en particulier pendant les deux guerres mondiales. L'automatisation a permis le développement de chiffrements plus complexes que ceux utilisant le papier et le stylo ou des dispositifs mécaniques très simples. Néanmoins, les machines devaient être efficaces et faciles à construire. Cela a conduit les développeurs à se diriger vers des designs qui répétaient un grand nombre de fois des opérations simples.

L'invention de la première machine cryptographique est attribuée à Jefferson pour son cylindre de chiffrement. De toutes ces machines inventées dans la première moitié du XXème siècle, nous en exposerons Enigma.

Enigma. La non moins notoire machine Enigma, utilisée par l'Allemagne, a été initialement conçue en 1918 à Berlin et présentée publiquement en 1923. Environ 100 000 machines Enigma ont été produites, dont environ 40 000 pendant la Seconde Guerre Mondiale. Le nom générique Enigma fait référence à une variété de modèles qui eux, sont basés soit sur le chiffre de Vigenère soit sur le chiffre de Beaufort.

La machine est composée d'un clavier, d'un réflecteur (soit un rotor d'inversion) [2] et, en général, de 3 rotors et d'un tableau de connexions. Le nombre de rotors peut aussi être de 5, 6 ou 7. Chaque rotor comporte 26 lettres et 3 autres caractères spéciaux dans certains cas.

Considérons le cas à 3 rotors. Dans le processus de chiffrement, au moment de choisir une lettre du texte clair L, le courant électrique traverse le premier rotor par la lettre L correspondante du texte clair, et les deux autres rotors effectuent les mêmes actions, mais avec des câblages différents. Ensuite, le réflecteur mappe le courant électrique à un autre endroit pour traverser les rotors en sens inverse à nouveau. L'ampoule allumé par le courant correspond à une lettre C qui sera le texte chiffré de L.

Le premier rotor tourne d'une position et, après 26 rotations, le second rotor tourne sur une position. De même, le troisième rotor tourne sur une position lorsque le second rotor effectue un tour complet. Le choix de l'ordre et de la position initiale des rotors ainsi qu'une permutation fixe de l'alphabet est le secret d'Enigma.

Etant donné son énorme espace de clé quotidienne ($\simeq 10^{22}$), il était admis qu'Enigma était assez sécurisée à des fins militaires ou diplomatiques.

Avant et pendant la deuxième Guerre Mondiale, casser Enigma s'est révélé être un défi de première importance. Les premiers résultats cryptanalytiques notables ont été obtenus par le mathématicien polonais Marjan Rejewski. C'est avec l'aide des clés quotidiennes découvertes par un employé du Bureau de Chiffrement Allemand que Rejewski a retrouvé la structure interne de la version à trois rotors d'Enigma. De 1932 à 1939, Rejewski ainsi que d'autres mathématiciens polonais ont permis d'importants progrès. Perfectionnement qui connaitra l'acmé avec le développement de la "Bombe" polonaise [3]. En 1938, les Allemands ont amélioré Enigma en lui ajoutant deux rotors. En 1939, les polonais fournissent leurs solutions ainsi que deux répliques d'Enigma aux Services Secrets Français et Britannique. Le Gouvernement Britannique créa le centre cryptologique Code and Cipher School (GC&CS) a Bletchley Park en Angleterre.

2. Le réflecteur connecte les sorties du dernier rotor par paires, redirigeant le courant dans les rotors selon un chemin différent. C'est ce réflecteur qui garantit le caractère involutif de d'Enigma: chiffrer est alors identique à déchiffrer. Cependant, le réflecteur empêche également Enigma de substituer une lettre à elle-même dans le texte chiffré.

3. La "Bombe" polonaise etait constituée de 6 machines Enigma qui travaillaient en parallèle afin d'obtenir les positions des rotors.

Alain Turing développe la "Bombe" britannique avec l'aide de Welchman, et c'est en Mai 1940 qu'elle devient opérationnelle. En 1941 et 1942, les cryptanalystes de l'armée américaine et de l'US Navy visitent Bletchley Park afin d'en apprendre davantage sur Enigma. A partir du mois d'avril 1943 jusqu'à la fin de la Guerre la "Bombe" américaine est utilisée.

Pour plus d'informations sur la cryptanalyse d'Enigma, nous renvoyons le lecteur à [GO03].

Pour une description plus détaillée des chiffres classiques, nous renvoyons le lecteur à [Kah96].

1.1.2 La cryptographie moderne

La Seconde Guerre Mondiale représente une charnière pour la cryptographie: les perspectives ont été rapidement et radicalement changées. Les fondements théoriques de la cryptographie se sont établis progressivement. Leurs assises aussi rigoureuses que scientifiques ont conduit à en établir un champ. Une spécialité à part entière.

La cryptographie ne peut être réduite à des communications secrètes, surtout de nos jours. Regardant attentivement l'état actuel de la cryptographie, nous devons prendre en considération les protocoles d'échange de clés secrètes, l'authentification des messages, les protocoles d'authentification, les signatures numériques, le vote électronique, les monnaies numériques (par exemple Bitcoin) ainsi de suite.

Dans l'ensemble, le rôle de la cryptographie moderne est de protéger chaque calcul distribué qui seraient susceptible de s'exposer à une vaste palette d'attaques.

Les principales différences de la cryptographie moderne sont à distinguer entre : les systèmes à clé secrète (ou symétrique), les systèmes à clé publique (ou asymétrique), les primitives sans clé (fonctions de hachage), les codes d'authentification de messages (schéma de MAC) et les signatures numériques. La cryptographie symétrique utilise la même clé pour le chiffrement et le déchiffrement, tandis que la cryptographie à clé publique utilise une clé publique pour le chiffrement et une clé privée connexe pour le déchiffrement.

Parties cryptographiques. Les personnages les plus communs de l'univers cryptographique moderne sont probablement Alice et Bob. Pour plus de flexibilité et pour faciliter la compréhension, au lieu de simplement utiliser A et B, les noms mentionnés ci-dessus sont préférés. Alice et Bob représentent les deux parties comprises dans un protocole. Entre ce couple pronominal émergera un troisième acteur : l'espion. Il sera baptisé Eve (eavesdropper).

Cryptographie symétrique

Pour les protocoles symétriques, nous pouvons considérer dans un scénario de base, qu'Alice et Bob veulent établir un secret commun et l'utiliser pour communiquer de manière chiffrée. Alice chiffre un message m en utilisant la clé déjà partagée, tandis que Bob déchiffre le texte chiffré (en utilisant la même clé) et récupère m lors de la réception. L'objectif des parties ayant convenu d'une clé secrète est d'empêcher Eve d'apprendre m, en supposant leur communication publique.

Notez que dans ce cadre, la même clé est utilisée pour chiffrer et déchiffrer. Ceci explique pourquoi ce protocole est appelé symétrique. Ici, la symétrie réside dans le fait que les deux parties partagent la même clé.

Une hypothèse en filigrane de tout système utilisant le chiffrement à clé symétrique est qu'Alice et Bob ont d'abord une certaine façon de partager une clé secrète. Dans les milieux militaires, ce n'est pas un problème car Alice et Bob peuvent se rencontrer physiquement dans un endroit sûr et convenir d'une clé. Cependant, dans le contexte virtuel d'Internet, les parties ne peuvent pas organiser une telle rencontre physique. Ceci est source de grandes préoccupations et limite effectivement l'utilité des méthodes à clé symétrique.

Il existe cependant d'autres paramètres où les méthodes à clé privée suffisent et sont largement étendues; un exemple est le chiffrement de disques durs, où l'utilisateur (à différents points dans le temps) utilise une clé secrète fixe pour à la fois écrire et lire sur le disque.

La syntaxe d'un système de chiffrement à clé symétrique. Les paragraphes qui suivent formalisent la description ci-dessus.

Un système de chiffrement à clé symétrique se compose de trois algorithmes probabilistes en temps polynomial (KEYGEN, ENCRYPT, DECRYPT) : de l'espace des clés $\mathcal{K}$, de l'espace des messages $\mathcal{M}$, de l'espace des textes chiffrés $\mathcal{C}$ correspondants. KEYGEN est un algorithme de génération de clés, ENCRYPT est une procédure pour chiffrer, DECRYPT est une procédure pour déchiffrer. La fonctionnalité de ces algorithmes est donnée à la Table 1.1.

Table 1.1: Algorithmes d'un système de chiffrement à clé symétrique.

KEYGEN	On note l'algorithme de génération de clés KEYGEN. KEYGEN est un algorithme probabiliste, dont la sortie est une clé sk, choisie conformément à une distribution donnée par le système de chiffrement respectif.
ENCRYPT	On note l'algorithme de chiffrement ENCRYPT. L'entrée d'ENCRYPT est une clé sk et un texte clair m. Sa sortie est constituée d'un texte chiffré c. Ainsi, le chiffrement de m avec la clé sk est désigné par $c = \text{ENCRYPT}(sk, m)$.
DECRYPT	On note l'algorithme de déchiffrement DECRYPT. L'entrée de DECRYPT est une clé sk et un texte chiffré c. Sa sortie est constituée d'un texte clair (message) m. Ainsi, le déchiffrement de c avec la clé sk est désigné par $m = \text{Decrypt}(sk, c)$.

Les deux principaux champs de la cryptographie à clé secrète sont les chiffrements par flux et les chiffrements par blocs. De courts rappels sur ces sujets sont exposés ci-dessous.

Les chiffrements par flux. Les chiffrements par flux sont attribués à Vernam (voir le chiffrement One Time Pad décrit dans la section 1.1.1). Ce qu'il a construit à l'époque est en substance, une machine électromécanique qui chiffrait automatiquement la communication par téléscripteur. Le texte clair était introduit dans la machine sous la forme d'une bande de papier, et le flux de clé avec une deuxième bande. Ce fut la première machine automatisant à la fois le chiffrement et la transmission.

Le mécanisme sous-jacent de chiffrements par flux consiste à chiffrer les bits individuellement. Un bit d'un flux de clé est ajouté modulo 2 à un bit de texte clair. Deux types de chiffrements par flux sont connus: les synchrones et les asynchrones. Dans le cas synchrone le flux de clé dépend uniquement de la clé. Dans le cas asynchrone le flux de clé dépend également du texte chiffré.

Une manière aisée, mais non sécurisée d'implémenter un chiffrement par flux [Ste87] est de le baser sur le registre à décalage: des registres à décalage à rétroaction linéaire (LFSR) ou des registres à décalage à rétroaction non-linéaire (NLFSR).

Contrairement au cas du chiffrement par bloc, la standardisation officielle n'était pas une priorité. Par conséquent, seule une compétition - secrète dirions-nous -, pour un choix pertinent de chiffrements par flux a été organisé et terminé en 2008. Parmi les finalistes, ont été retenu : Salsa20/12 [Ber08], Rabbit [BVP+03], HC-128 [Wu08] et SOSEMANUK [BBC+08] pour le logiciel et Grain v1 [HJM07], MICKEY v2 [BD08] et Trivium [DCP06] pour le matériel.

Les chiffrements par bloc. Actuellement, les constructions les plus connues dans la cryptographie à clé secrète sont les chiffrements par blocs. En 1977, la proposition de chiffrement à clé symétrique d'IBM, appelé Lucifer, a été choisie comme le chiffrement principal par bloc. C'est désormais le Data Encryption Standard (DES). En réponse aux attaques qui sont apparues [BS91, Fou98, KPP+06], il a été montré que le double DES peut être attaqué avec une complexité de 2^{57} en temps et 2^{56} en mémoire [MH81]. Le triple DES [BB12] est donc devenu le principal mode de chiffrement de l'industrie.

La nécessité d'avoir des textes clairs et des clés de plus en plus longues a mené au remplacement progressif du triple DES par l'Advanced Encryption Standard (AES) [NIS01] aussi appelé Rijndael [DR98]. Outre la sécurité, la vitesse était la caractéristique la plus importante lors du choix de Rijndael comme le nouveau standard.

Alors que les chiffrements par flux doivent développer la clé, les chiffrements par blocs utilisent la même clé pour effectuer le chiffrement d'un bloc entier de bits de textes clairs. Par conséquent, le chiffrement

de n'importe quel bit de texte clair dans un bloc donné dépend de tous les autres bits de texte clair dans ce même bloc. Ce fait est étroitement lié aux notions de confusion [4] et de diffusion [5], initialement définies par Shannon [Sha49].

La taille usuelle des blocs dans les applications cryptographiques est de 128 bits.

Modes d'opération. Choisir un mode d'opération [6] approprié est essentiel pour les chiffrements par blocs. Cet outil peut affecter la vitesse des processus de chiffrement et de déchiffrement, la sécurité contre les adversaires actifs, contre les adversaires passifs et la propagation d'éventuelles erreurs.

Les cinq modes confidentiels d'opération normalisés par le NIST [Dwo01] sont ECB, CBC, CFB, OFB, CTR.

En 2010, le NIST a approuvé XTS [Dwo10] comme un mode confidentiel d'opération conçu pour les périphériques de stockage. XTS est une variante du mode XEX [7].

Selon l'application cryptographique sous-jacente, l'un de ces modes d'opération est utilisé. Chacun de ces modes a des avantages et des inconvénients, mais l'ECB est l'un des plus indésirables étant donné qu'il préserve certains motifs lors de l'encryption.

Selon l'application cryptographique sous-jacente, l'un de ces modes d'opération est utilisé. Chacun de ces modes a des avantages et des inconvénients, mais l'ECB est l'un des plus indésirables étant donné qu'il préserve certains motifs lors de l'encryption.

Cryptographie à clé publique

Comme l'a noté Diffie dans [Dif88], la cryptographie à clé publique est apparue en 1975 [DH76]. Cependant, Merkle [Mer] avait une proposition sérieuse dans cette perspective, depuis l'automne de 1974 [8]. Connue comme "le puzzle de Merkle", la technique a été littéralement basée sur des "puzzles" qui étaient plus faciles à résoudre pour l'émetteur et le récepteur que pour un adversaire.

Annonçant une "révolution en cryptographie", le document de Diffie et Hellman publié en 1976 [DH76] était le premier document à présenter les fondements théoriques de la cryptographie à clé publique. Ainsi, ils ont défini le concept de système cryptographique à clé publique et ont discuté des notions auxiliaires, mais nécessaires telles que *les fonctions à sens unique* et *trappe*.

Après cette percée majeure, l'année 1978 aura révélé deux systèmes de chiffrement à clé publique d'une importance nodale : RSA - qui était une création commune de Rivest, Shamir et Adleman [RSA78] et le système du sac a dos de Merkle-Hellman [MH78]. Avec les systèmes cryptographiques à clé publique émergents, le problème des signatures numériques est aussi apparu. Nous developperons ces sujets dans la section 3.3.

Le système cryptographique RSA est fondé sur la difficulté de factoriser un grand nombre, tandis que le régime Merkle-Hellman tenait pour hypothèse de sécurité sous-jacente l'une des versions du problème du sac à dos.

RSA a passé le test du temps restant sécurisé (modulant le changement de la longueur recommandée de la clé, l'adoption d'un padding adéquat, en évitant les attaques par diffusion, etc.), mais le système cryptographique de Merkle-Hellman a été cassé par Shamir quelques années après sa découverte [Sha84].

Outre l'introduction des fondements théoriques de la cryptographie à clé publique, Diffie et Hellman proposent un protocole d'échange de clés basé sur la difficulté du problème du logarithme discret (DLP, détaillé dans la section 2.1.2) [DH76]. Leur protocole a été implémenté plus tard à base d'EC. Il faut noter un autre système de chiffrement lié à DLP introduit par El Gamal [EG84].

4. La relation entre la clé et le texte chiffré doit être complexe. Cela veut dire que chaque bit de texte chiffre doit dépendre de tous les bits de la clé.
5. Dans un bon système de chiffrement, changer un bit de texte clair changera en moyenne la moitier des bits du texte chiffré.
6. D'appres [Dwo01] un *mode d'operation* est *un algorithme pour la transformation cryptographique de données qui fait intervenir un algorithme de chiffrement symmetrique par bloc.*
7. Développé par Rogaway et présenté dans [Rog04a]
8. Merkle a suivi le cours CS244 donné par Hoffman à Berkeley.

Dans un algorithme asymétrique, chaque utilisateur sélectionne une paire de clés constituée d'une clé privée sk et d'une clé publique pk. L'utilisateur doit garder le secret de sk. sk et pk sont liées par une *fonction à sens unique*, un concept défini ci-dessous.

Les systèmes cryptographiques à clé publique sont généralement plus lents que ceux à clé secrète. Par conséquent, le chiffrement asymétrique est le plus couramment utilisé dans la pratique pour les transports de clé sécurisé (l'emballage des clés [9].). Ensuite, la clé partagée est utilisée pour le chiffrement des données au moyen d'algorithmes symétriques.

La cryptographie à clé publique a été en réalité découverte en 1969 par Ellis lorsqu'il travaillait pour le British Government Communications Headquarters. L'échange de clé de Diffie-Hellman et le système RSA ont été découverts indépendamment par GCHQ, bien des années avant que le monde de la cryptographie ne les découvrent [Sin99]. Ces faits ont été connus et communiqués récemment.

Definition 1.1 *Soit $\{0,1\}^*$ l'ensemble des chaînes de caractères binaires et $f : \{0,1\}^* \to \{0,1\}^*$. f est appelée une* fonction a sens unique *si*

1. *Il existe un algorithme efficace qui prend x en argument et retourne $f(x)$*
2. *Pour tout adversaire en temps polynomiale $\mathcal{A}$, la probabilité suivante est négligeable en κ:*

$$Pr[x \xleftarrow{\$} \{0,1\}^{\kappa}; f(\mathcal{A}(f(x,y)))f(y) = f(x)].$$

Définition d'un système de chiffrement à clé publique. Un système de chiffrement à clé asymétrique est constituée de quatre algorithmes probabilistes en temps polynomial (SETUP, KEYGEN, ENCRYPT, DECRYPT), un espace des clés $\mathcal{K}$, un espace des messages $\mathcal{M}$ et un espace des textes chiffrés $\mathcal{C}$. SETUP est une procédure pour générer les paramètres du système, KEYGEN est un algorithme de génération de clés, ENCRYPT est une procédure pour chiffrer et DECRYPT est une procédure pour dechiffrer. La Table 2.2 décrit ces algorithmes.

Table 1.2: Algorithmes d'un système de chiffrement à clé publique.

SETUP	Soit κ le paramètre de sécurité. SETUP(1^{κ}) génère les paramètres globaux du système.
KEYGEN	Le rôle de KEYGEN est de générer la clé de chiffrement publique pk et la clé de déchiffrement privée correspondante sk.
ENCRYPT	Soit le texte clair m. Le rôle d'ENCRYPT est de chiffrer m pour obtenir un texte chiffré $c = \text{Encrypt}(pk, m)$.
DECRYPT	Le but de DECRYPT est de déchiffrer le texte chiffré c en utilisant la clé de déchiffrement sk. La sortie peut être soit le texte clair m ou $\perp$, qui est un symbole d'invalidité.

1.2 Résumé de la thèse

Nous présentons les fondements théoriques sous-jacents liés à nos résultats dans le chapitre 3.

Nous fournissons une courte introduction sur les fonctions de hachage et les codes d'authentification de message (MAC) dans le chapitre 3. Nous présentons également le chiffrement authentifié (à la fois le générique composition paradigme [BN08] [10] et des solutions dédiées [11] [Jut01, GD01, RBBK01]) donne un aperçu des systèmes de signature numérique. Les courbes elliptiques sont introduits et leur rôle dans la cryptographie est discuté, au vu des résultats de cryptanalyse présentés dans la section 5.6.

Nos contributions originales sont structurées en deux chapitres: la conception de protocoles (chapitre 4) et des algorithmes pour la cryptographie intégrée (chapitre 5).

Nous présentons un protocole de co-signature prouvé sûr et un système de chiffrement authentifié prouvé sûr dans le chapitre 4. Le chapitre 4 comprend également un protocole distribué de type Fiat-Shamir permettant l'authentification sur des réseaux.

9. L'idée de l'emballage de clé a été proposée pour la première fois dans le papier introduisant RSA [RSA78]
10. La composition paradigme générique se réfère à la combinaison d'un système de chiffrement et un MAC.
11. Une solution dédiée se réfère à la fourniture à la fois de la confidentialité et d'authenticité dans dans le même système.

La plupart des résultats présentés dans le chapitre 5 concernent des améliorations algorithmiques. Nous décrivons ces améliorations du point de vue de la cryptographie à bas coût.

La *cryptographie à bas coût* est adapté pour les appareils contraintes dans lequel les concepteurs doivent trouver des compromis entre la performance, la sécurité et le coût. Les contraintes peuvent être la puissance de calcul, mémoire, bande passante, ou de la sécurité.

La section 5.2 comprend un nouveau procédé qui permet de doubler la vitesse de l'algorithme de Barrett à l'aide de modules composites spécifiques. Nous décrivons un nouvel algorithme de multiplication basé sur le retour sur trace dans la section 5.4. Cet algorithme s'avère particulièrement adapté aux microprocesseurs bon marché. présente des nouvelles stratégies d'accélération de codes BCH utilisant des versions polynomiales de l'algorithme de Barrett dans la section 5.3.1. Nous décrivons un nouveau code correcteur d'erreurs inspiré par le système de chiffrement Naccache-Stern [NS97, CMNS08] dans la section 5.3.2. Notre contribution présenté dans la section 5.5 permet de régulariser le débit d'extracteurs de von Neumann.

Finalement, nous présentons une nouvelle attaque en fautes sur les algorithmes de signature à courbes elliptiques, à partir de [NSS04] dans la section 5.6.

Donc, les résultats du manuscrit tournent autour de: la conception du protocole, des améliorations algorithmiques et des attaques.

CHAPTER 2

INTRODUCTION

If you reveal your secrets to the wind, you should not blame the wind for revealing them to the trees.
Kahlil Gibran.

Summary

This chapter introduces terminology and explains the role of cryptography within the broader field of cryptology. We discuss milestones, and overview the field's evolution as a short journey starting with pre-computer (Section 2.1.1) and reaching post-quantum ciphers (Section 2.1.2). We provide technical details of symmetric and asymmetric key cryptography (Section 2.1.2), discuss computational security, complexity theory and hardness assumptions, and recall the security notions for secret-key and public-key cryptography.

Section 2.2 describes the topics addressed in this thesis and sets the goals to be achieved. The main results in this thesis are new protocols, authenticated encryption and signature schemes, error correcting code ideas, as well as cryptanalysis and algorithmic improvements.

In the protocol and scheme design part of this thesis, we describe a provably secure co-signature protocol, introducing a novel form of fairness (*legal fairness*), as well as a provably secure authenticated encryption scheme called Offset Merkle-Damgård (OMD). OMD is a keyed compression function mode of operation. Also, we present a lightweight distributed Fiat-Shamir authentication protocol that enables network authentication. Protocol and scheme design contributions are further addressed in Chapter 4.

One of our algorithmic improvements is a new backtracking-based multiplication algorithm, particularly suited to lightweight microprocessors when one of the operands is known in advance. Other improvements are a method allowing to double the speed of Barrett's algorithm by using specific composite moduli, new BCH speed-up strategies using polynomial versions of Barrett's method as well as a new error-correcting code (ECC) inspired by the Naccache-Stern cryptosystem. We present a new method to streamline the pace of random bits output by a von Neumann extractor and, finally, venture into cryptanalysis and describe a new fault attack on elliptic curve cryptography (ECC) implementations. Algorithmic improvements and cryptanalysis results appear in Chapter 5.

Our main contributions are listed in Section 2.3 along with their publication references and abstracts.

2.1 A Brief Introduction to the History of Cryptography

According to the Merriam-Webster dictionary [MWb], the term *cryptology* was first used in 1935 and refers to *the scientific study of cryptography and cryptanalysis*. The word *cryptography* was first mentioned in 1658 and its origin is closely related with the Modern Latin concept *cryptographia* inherited from the Greek terms *kryptos* (hidden, secret) and *graphein* (to write). The first use of the word *cryptanalysis* dates back to 1923 [MWa].

Cryptography's core role is creating systems for keeping information secret and unaltered. In parallel, cryptanalysis arose as the art of breaking the ciphers designed by cryptographers.

Cryptography's main goals are the enforcement of *confidentiality* (keeping information secret), *data authentication* (the assurance of message non-alteration during transmission), *entity authentication* (knowing who sent a message) and *non-repudiation* (preventing the denial of past actions). The generic term *cryptosystem* refers to algorithms achieving some or all of the previously mentioned goals.

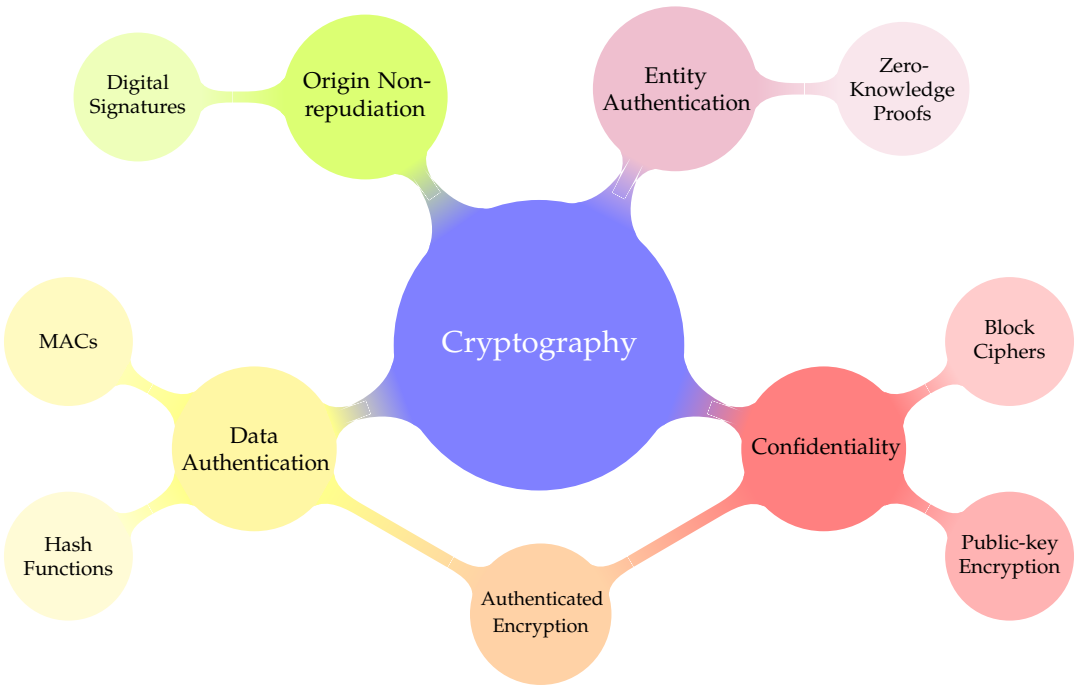


Figure 2.1: Some cryptographic goals and primitives.

The concern of keeping secrets is millennia old. During centuries the historical users of encryption were diplomats, spies and militaries.

The recent explosion of telecommunications made cryptography a general necessity.

Notations. The following notations will be used within this thesis:
- $\mathcal{M} = \{m | m \in \texttt{Alph}_\texttt{m}^*\}$ will denote the non-empty set of plaintext space (also called *message space* or *cleartext space*), where $\texttt{Alph}_m$ is a non-empty alphabet.
- $\mathcal{C} = \{c | c \in \texttt{Alph}_\texttt{c}^*\}$ will denote the set of ciphertexts (also called *ciphertext space*), where $\texttt{Alph}_c$ is a non-empty alphabet.
- $\mathcal{K}$ will denote the set of keys. $\mathcal{K}$ is also called *key space*.

Usually, $\texttt{Alph}_m = \texttt{Alph}_c$.

Let S be a finite set. We denote by $x \xleftarrow{\$} S$ or $x \in_R S$ the operation of picking an element uniformly from S.

If α is neither an algorithm nor a set then $x \leftarrow \alpha$ will denote a simple assignment statement.

2.1.1 Pre-Computer Cryptography

Classical Ciphers

Classical ciphers are historically important. This category of ciphers is often referred to as "pen and paper ciphers" and actually includes simple cryptographic algorithms that can be computed directly by hand or using very simple mechanical tools.

One of the most ancient encryption device examples is the *Scytale* cipher, originated in Sparta, Greece. The device *Scytale* was a cylinder and a parchment ribbon wounded around it, on which the message was written lengthwise. Popular in the 6th century B.C., *Scytales* belong to the category of transposition ciphers.

The CÆSAR **Cipher.** Named after its most famous user, Julius Cæsar, this cipher dates from the 1st century B.C. The CÆSAR encryption system is a single alphabet substitution cipher.

Consider the 26-letters alphabet, *i.e.* $\texttt{Alph}_m^* = \texttt{Alph}_c^* = \mathbb{Z}_{26}$. The CÆSAR cipher cyclically shifts each plaintext letter by 3 positions to obtain the ciphertext. The number of shifts can be replaced by any encryption key $k \in \mathbb{Z}_{26}$. Decryption is a reverse shift.

A CÆSAR encryption disk is shown in Figure 2.2.

Figure 2.2: CÆSAR encryption disk ($k = 10$).

The VIGENÈRE **Cipher.** The VIGENÈRE cipher was developed by Blaise de Vigenère in the 16th century [Kah96].

Again, consider the 26-letters alphabet and chose a key κ of length r. The VIGENÈRE cipher can be seen as an encryption system constructed from r different CÆSAR ciphers (corresponding to each character of κ). If the message to be encrypted is longer than κ, the key is used periodically.

Hence, the VIGENÈRE cipher is a poly-alphabetic substitution system.

The ONE TIME PAD**.** Also known as the Vernam cipher can be easily described as a VIGENÈRE cipher whose key length equals the plaintext length. Also, all keys must be only used once.

ONE TIME PAD, illustrates the fact that *random number generators* play a crucial role in cryptography.

If perfectly implemented (using unique, random keys, at least as long as the messages), the One Time Pad is unconditionally secure (we refer the reader to Section 2.1.2).

Transposition Ciphers. Another type of ciphers are the *transposition ciphers*. To have a clear example we consider a columnar transposition cipher. Such a transposition cipher starts by dividing the message

S	E	C	R	E	T
E	X	P	L	A	I
N	T	R	A	N	S
P	O	S	I	T	I
O	N	C	I	P	H
E	R	S	X	X	X

C	E	E	R	S	T
P	X	A	L	E	I
R	T	N	A	N	S
S	O	T	I	P	I
C	N	P	I	O	H
S	R	X	X	E	X

Figure 2.3: Columnar transposition cipher example.

m into blocks of ℓ bits, writing it row-wise in a table of given size and applying a fixed permutation σ to the message [1]. The obtained ciphertext is read out column-wise.

In Figure 2.3 we give an example in which m is `EXPLAINTRANSPOSITIONCIPHERS`, κ is `SECRET` and the padding character is X. The permutation is given by the alphabetical order of κ's letters, resulting in $\sigma = (3, 2, 5, 4, 1, 6)$. We obtain the ciphertext `PRSCSXTONRANTPXLAIIXENPOEISIHX`.

Electrical and Mechanical Ciphers

To automate encryption and decryption, different mechanical designs were proposed, especially during the World Wars. Automation allowed ciphers to become much more complex than paper and pen codes or simple mechanical devices. Nonetheless, machines had to remain efficient and easy to construct. This led developers to adopt designs that repeatedly perform simple operations.

The first cryptographic machine is credited to Jefferson who invented his ciphering cylinder [Kah96]. Amongst the numerous electromechanical machines invented during the 1900s two designs are particularly famous: HAGELIN's ciphering machines [Kah96] and ENIGMA [Kah96].

HAGELIN's Ciphering Machines. Boris Hagelin was a Swedish engineer who invented the C-35 [2], C-38 (M-209) and C-52 ciphering machines. During the second World War about 140.000 C-38 machines were produced. HAGELIN's machines were a trade off between security and efficiency - designed for tactical purposes.

C-38 is based on 6 rotors with 26, 25, 23, 21, 19 and 17 pins. Pins can be either active or passive. When encrypting a letter, all rotors turn by one position. Hence, after 26 encryptions the first rotor will get back to its initial place (for the sixth rotor this takes only 17 encryptions).

HAGELIN's ciphering machines can thus be regarded as a mechanical versions of the VIGENÈRE cipher. Because the number of pins on the rotors are co-prime. The device's period is $26 \times 25 \times 23 \times 21 \times 19 \times 17 = 101,405,850$.

C-52 was an improved model, one of the last electromechanical cipher machines before the computer age.

ENIGMA. The famous ENIGMA machine, used by Germany, was designed in 1918 in Berlin and publicly presented in 1923. About 100.000 ENIGMA machines were produced, of which around 40.000 machines during World War II. The generic name ENIGMA actually refers to a variety of models which are based on either VIGENÈRE or BEAUFORT ciphers.

ENIGMA's components are a keyboard, a reflector (*i.e.* reversal rotor) [3] and, usually, 3 rotors and a plug-board. The number of rotors can be 5, 6, or 7. Each rotor features 26 letters plus 3 other special characters in some cases.

In the 3-rotor variant encryption process, when pressing a plaintext letter switch L, an electronic current enters the first rotor at the plaintext corresponding to L and the other two rotors perform the same

1. Depending on a key κ.
2. The corresponding patent was published in 1937.
3. The reflector connected outputs of the last rotor in pairs, redirecting current back through the rotors by a different route. The reflector ensured that Enigma is an involution: conveniently, encryption was the same as decryption. However, the reflector also gave Enigma the property that no letter ever encrypted to itself.

actions, but with distinct wirings. Then, the reflector maps the electrical current to another place to reversely pass through the rotors again. The panel's light bulb corresponding to a letter C lightened up by the current is the ciphertext of L.

The first rotor will turn by one position and, after 26 rotations, the second rotor will turn by one position. Similarly, the third rotor will rotate by a position when the second rotor completes an entire revolution. ENIGMA's secret key is the choice, order and initial position of the rotors plus a fixed alphabet permutation.

Given its huge daily keyspace ($\simeq 10^{22}$), ENIGMA was thought to be secure enough to serve military and diplomatic purposes.

An ENIGMA machine is presented in Figure 2.4.

Figure 2.4: A four-rotor ENIGMA machine.

Before and during World War II, breaking ENIGMA was a prime importance challenge. First notable cryptanalytic results were obtained by the Polish mathematician Marjan Rejewski. With the aid of daily keys table received from an employee of the German Cipher Bureau, Rejewski reconstructed the internal structure of ENIGMA's three rotors. During 1932-1939, Rejewski and other Polish mathematicians made steady progress that climaxed with the development of the Polish "Bomba" [4]. In 1938, Germans enhanced ENIGMA with two further rotors. In 1939 the Poles provided their solutions and two ENIGMA replicas to the French and British Intelligence Services. A cryptologic center is created by the British Government Code and Cipher School (GC&CS) in Bletchley Park, England. Alan Turing develops a British "Bombe" with the aid of Welchman. The "Bombe" started being used in May 1940. In 1941 and 1942, cryptanalysts of the US Army and the US Navy visit Bletchley Park to learn about ENIGMA. Since April 1943 and until the end of the war the American "Bombe" was used.

For further information about the cryptanalysis of ENIGMA the reader may consult [GO03].

For a more detailed description of classical ciphers we refer the reader to [Kah96].

2.1.2 Modern Cryptography

World War II represented a turning point for cryptography: perspectives changed quickly and drastically. The theoretical foundations of cryptography were progressively established, leading to the rigorous, scientific study of this field.

Cryptography cannot be reduced to secret communications only, especially nowadays. Looking attentively at the current status of cryptography, we have to take into consideration key exchange protocols,

4. The Polish "Bomba" consisted of six ENIGMA machines working in parallel to obtain the rotors' positions.

message authentication, entity authentication protocols, digital signatures, electronic voting, digital coins (*e.g.* Bitcoin) and so on.

All in all, modern cryptography's role is to protect each and every distributed computation that could be prone to a vast palette of attacks.

The main dissociation in modern cryptography is between secret (or symmetric) key systems, public (or asymmetric) key systems, unkeyed primitives (hash functions), message authentication code algorithms (MAC algorithms) and digital signatures. Symmetric cryptography uses the same key both for encryption and decryption, while public key cryptography uses a public key for encryption and a related private one for decryption.

Cryptographic Parties. For more flexibility and to ease understanding, instead of simply using A and B, we call parties Alice and Bob. Eve usually stands for eavesdropper.

Symmetric Cryptography

In a symmetric-key setting, Alice and Bob need to establish a common secret and use it to communicate privately. Alice encrypts a message m using the already shared key, while Bob decrypts the ciphertext (using the same key) and recovers m. The purpose of the parties is to prevent Eve from learning m, as their communication is assumed to take place over a public channel.

In this setting, the same key is used to encrypt and decrypt. This explains why this setting is called *symmetric*. Here, symmetry lies in the fact that both parties use the same key.

An implicit assumption in any system using symmetric-key encryption is that Alice and Bob have some way to initially share a key secretly. In military settings, Alice and Bob may physically meet in a secure location to agree upon a key. In the Internet era, however, parties cannot always arrange any such physical meeting, but this issue is solved in a different manner (*e.g.* by means of hybrid encryption).

There are many settings where private-key methods are in wide use; one example is disk encryption, where the user (at different points in time) uses a fixed secret key to both write to and read from the disk.

The Syntax of a Symmetric Key Encryption Scheme. A symmetric-key encryption scheme consists of three probabilistic polynomial-time[5] (PPT) algorithms (KEYGEN, ENCRYPT, DECRYPT) and the corresponding key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. KEYGEN is an algorithm for generating keys, ENCRYPT is a procedure for encrypting, and DECRYPT is a procedure for decrypting. The functionality of these algorithms is given in Table 2.1.

Table 2.1: Algorithms of a symmetric-key encryption scheme.

KEYGEN	KEYGEN is the key generation algorithm. KEYGEN is a probabilistic algorithm whose output is a key sk, chosen in accordance with a distribution given by the encryption scheme.
ENCRYPT	ENCRYPT denotes the encryption algorithm[6]. The input of ENCRYPT is a key sk and a plaintext m. ENCRYPT's output is a ciphertext c. Thus, the encryption of m with the key sk is referred to as $c =$ENCRYPT(sk, m).
DECRYPT	DECRYPT denotes the decryption algorithm. The input of DECRYPT is a key sk and a ciphertext c. DECRYPT outputs a plaintext (message) m. Hence, the decryption of c with the key sk is referred to as $m =$DECRYPT(sk, c).

The two main sub-areas of secret key cryptography are *stream ciphers* and *block ciphers*. Very short reminders on these sub-areas follow.

5. in the security parameter
6. Encryption may be randomized or stateful.

Stream Ciphers. Stream ciphers are attributed to Vernam [7] [Ver26] (see the ONE TIME PAD cipher listed in Section 2.1.1). Vernam built an electromechanical machine that automatically encrypted teletypewriter communication. The plaintext was fed into the machine as a paper tape, and the key stream as a second tape. That was the first time at which encryption and transmission were automated and merged into one machine.

Stream ciphers usually encrypt bits individually [8]. A bit of a key stream is XORed with a plaintext bit. Two stream cipher types are known: synchronous, and asynchronous. In synchronous stream ciphers the key stream depends only on the key. In asynchronous stream ciphers the key stream depends on the ciphertext too.

An easy way to implement (unsafe) stream-ciphers [Ste87] consists in using linear feedback shift registers (LFSRs) and non-linear feedback shift registers (NLFSRs). Despite their insecurity as primitives by their own rights LFSRs and NLFSRs [9] are sometimes used as inner design parts.

Unlike the case of block ciphers, stream-cipher standardization was not a priority. Therefore, only a slightly visible competition for choosing good stream ciphers was organized in 2008 [Conb]. Among the finalists were Salsa20/12 [Ber08], Rabbit [BVP+03], HC-128 [Wu08] and SOSEMANUK [BBC+08] (software-based stream ciphers) and Grain v1 [HJM07], MICKEY v2 [BD08] and Trivium [DCP06] (hardware-based stream ciphers).

Block Ciphers. Block ciphers are the core building blocks of secret key cryptography. In 1977 the symmetric key cipher Lucifer was chosen by the NIST as the Data Encryption Standard (DES) [NIS99]. As a response to the complexity trade-offs and attacks on DES that appeared [BS91, Fou98, KPP+06] [10]. Triple DES [BB12] became the industry's prime encryption mode.

The need for longer plaintexts and longer keys eventually led to the adoption of the Advanced Encryption Standard (AES) [NIS01] originally called Rijndael [DR98]. Apart security, speed was the most important reason for adopting Rijndael as the new standard.

While stream ciphers need to expand the key, block ciphers use the same key to encrypt an entire plaintext block. Hence, the encryption of any given plaintext bit in a block depends on every other plaintext bit in the same block. This fact is closely related with the notions of *diffusion* [11] and *confusion* [12], defined by Shannon in [Sha49]. The usual block length in cryptographic applications is 128 bits [13].

Modes of Operation. Choosing an appropriate *mode of operation* [14] is critical for block ciphers. The mode of operation may affect the encryption and decryption speed, security against passive and active adversaries as well as the the propagation of possible errors.

The five confidentiality (*i.e.* encryption) modes of operation standardised by NIST [Dwo01] are schematically described in Figures 2.5 to 2.8 and 2.9. In 2010, NIST approved XTS [Dwo10] as a confidentiality mode of operation designed for storage devices. XTS is a variation of XEX mode [15].

Encryption and decryption functions follow the color codes given in Table 2.1. We use abbreviations to refer to the functions. Plaintexts and ciphertexts are assumed to be already divided into blocks. For clarity, we will only show the first three blocks in each diagram.

7. Recent work by Bellovin seems to indicate that Vernam may not have invented them [Bel11], but he certainly was the first to build a practical device using them.
8. However, *e.g.* RC4 [MvOV96] encryption is performed bytewise
9. NLFSRs are vulnerable to distinguishing attacks *e.g.* linear approximations, short period. Various papers on the cryptanalysis of Grain were based on some NLFSR vulnerabilities [BGM06, ZW09].
10. Merkle and Hellman have shown that double DES can be attacked with 2^{57} encryptions and 2^{56} 14-byte values space [MH81], while van Oorschot and Wiener have reported an attack that requires 2^{72} encryptions and 16 GB storage [VOW96a].
11. In a good cipher, flipping a bit in the plaintext causes flipping of *circa* 50% of the ciphertext's bits.
12. The relations hip between the key and the ciphertext must be complex, meaning that each ciphertext bit must depend on all the key bits.
13. although the block length of DES which is still used by the banks has is 64 bits
14. According to [Dwo01], a *mode of operation* is "an algorithm for the cryptographic transformation of data that features a symmetric key block cipher algorithm".
15. Developed by Rogaway and presented in [Rog04a].

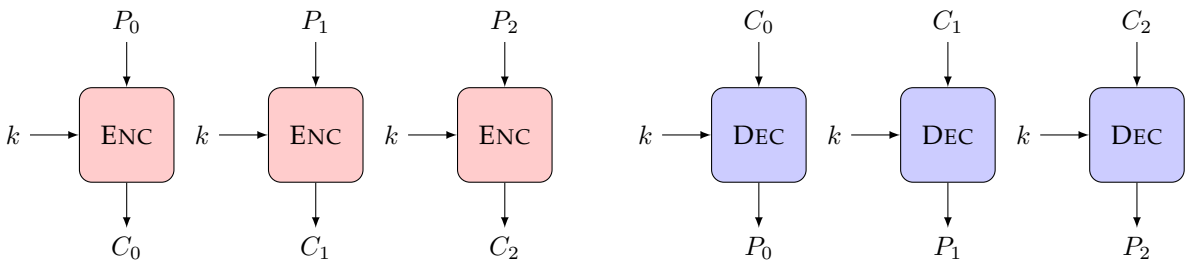


Figure 2.5: The ECB mode of operation - ENCRYPT and DECRYPT algorithms.

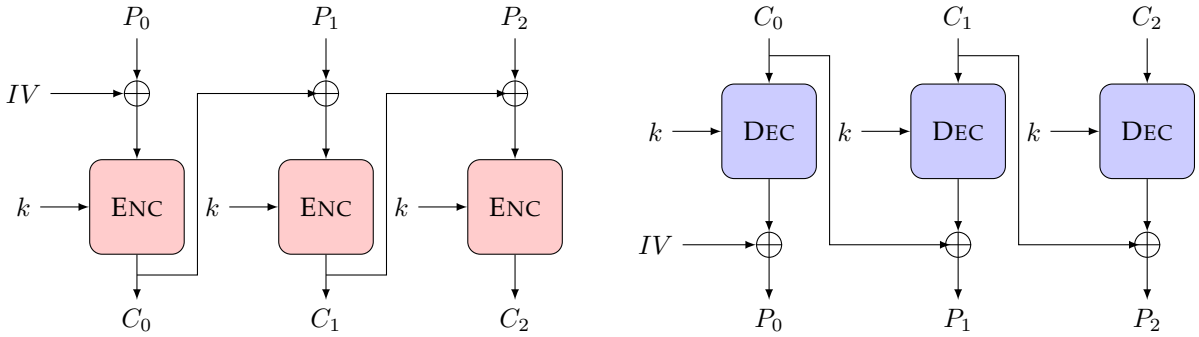


Figure 2.6: The CBC mode of operation - ENCRYPT and DECRYPT algorithms.

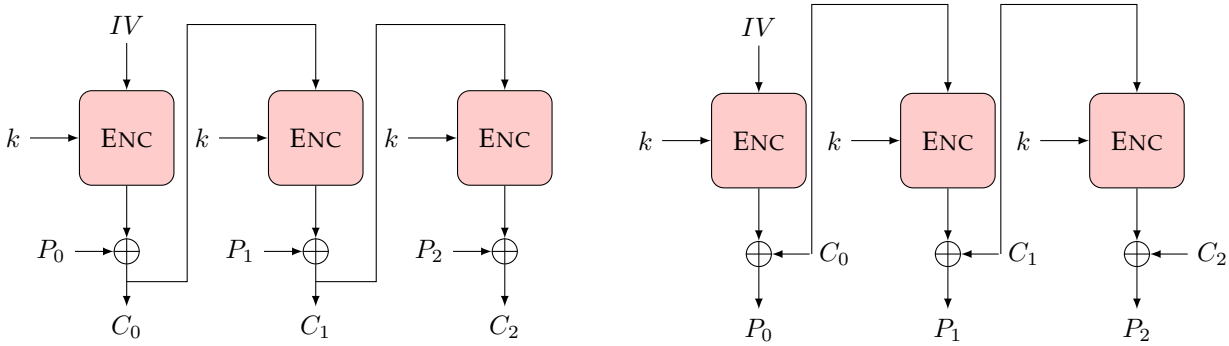


Figure 2.7: The CFB mode of operation - ENCRYPT and DECRYPT algorithms.

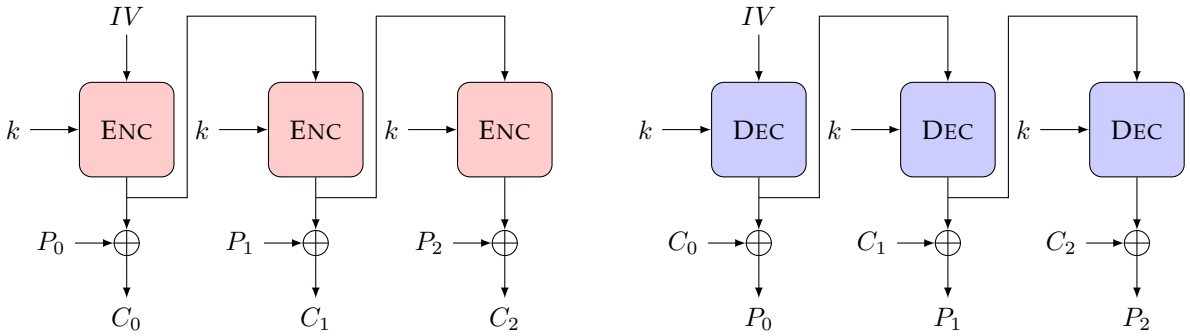


Figure 2.8: The OFB mode of operation - ENCRYPT and DECRYPT algorithms.

Depending on the underlying cryptographic application, one of these modes of operation is used. Each of the five modes has pros and cons, but the pattern preserving issue[16] of ECB is one of the most undesirable ones. To prevent pattern preservation and to hide repetitions, encryption modes CBC, CFB, and OFB modes include a data block named initialization vector (IV). The IV is used in an initial step of

16. Identical plaintext blocks are encrypted into identical ciphertext blocks.

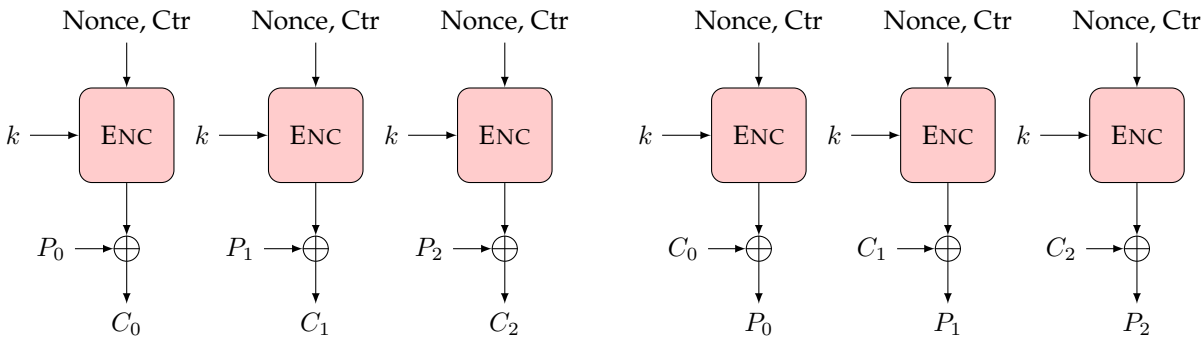


Figure 2.9: The CTR mode of operation - ENCRYPT and DECRYPT algorithms.

message encryption in the corresponding decryption process. The IV need not be secret. In CBC and CFB, the IV for any particular execution of encryption must be unpredictable. In OFB, unique IVs (also called "fresh") must be used for each encryption. CTR uses a *counter* instead of a traditional IV.

Public-Key Cryptography

As noted by Diffie in [Dif88], public key cryptography officially appeared in 1976 [DH76]. A year earlier Merkle [Mer] had a seminal proposal in this direction [17]. Known as "Merkle's puzzle", the technique was based on problems which are easier to solve by the sender and receiver who cooperate than by the adversary.

Announcing a "revolution in cryptography", Diffie and Hellman's 1976 paper published in 1976 [DH76] was the first to lay the theoretical foundations of public key cryptography. [DH76] defined the concept of public key cryptosystem and discussed auxiliary but necessary notions such as *one-way functions* and *trapdoors*.

After this major breakthrough, 1978 brought up two extremely important public key encryption schemes: RSA, by Rivest, Shamir and Adleman [RSA78] and the Merkle-Hellman knapsack [MH78]. Along the emerging public key cryptosystems, digital signatures also started arising. We will develop this topic in detail in Section 3.3.

RSA is based on the hardness of the integer factorization problem, while Merkle-Hellman's underlying security assumption is a specific knapsack problem.

RSA passed the test of time remaining secure (modulo changing the recommended length of the key, adopting proper padding, avoiding broadcast attacks, etc.), but the Merkle-Hellman cryptosystem was broken by Shamir a few years after its discovery [Sha84].

Besides introducing the theoretical foundations of public key cryptography, Diffie and Hellman proposed a key exchange protocol based on the hardness of the Discrete Logarithm Problem (DLP, see Section 2.1.2) [DH76]. Later, elliptic curve versions of Diffie-Hellman's protocol were developed. Another DLP-related cryptosystem worth of mentioning in the introduction is ElGamal [EG84].

In an asymmetric setting each user selects a key pair consisting of a private key sk and a corresponding public key pk. The user must keep sk secret. sk and pk are related by a *one-way function*, a concept defined next.

Public-key cryptosystems are usually slower than secret-key ones. Hence, asymmetric encryption is most commonly used in practice for secure symmetric key transport (key wrapping [18]) and then the transported key is used for data encryption by means of symmetric algorithms.

Public-key cryptography was actually discovered in 1969 by Ellis while working for the British Government Communications Headquarters. The Diffie-Hellman key exchange and the RSA were hence independently discovered by GCQH, years before their development by the cryptographic world [Sin99]. These facts surfaced only in 1997.

17. Merkle enrolled in CS244, the course offered by Hoffman at Berkeley.
18. The idea of key wrapping was proposed in the original RSA paper [RSA78].

Concrete Security Conventions. The advantage Adv of an adversary $\mathcal{A}$ is a measure of how successful an attack against an algorithm A_1 is, by distinguishing it from an idealized version of A_1 (*i.e.* a version in which the security property of A_1 clearly holds.).

As usual with the concrete-security definitions, we use the resource parametrized function $\mathrm{Adv}^{\mathrm{xxx}}_{\Pi}(r)$ to denote the maximal value of the adversarial advantage (*i.e.* $\mathrm{Adv}^{\mathrm{xxx}}_{\Pi}(r) = \max_{\mathcal{A}}\{\mathrm{Adv}^{\mathrm{xxx}}_{\Pi}(\mathcal{A})\}$) over all adversaries $\mathcal{A}$, against the xxx property of a primitive or scheme Π, that use resources bounded by r. The resource parameter r, depending on the notion, may include time complexity (t), length of queries and number of queries that an adversary makes to its oracles. If a resource parameter is irrelevant in the context then we omit r; *e.g.* for information-theoretic security bounds the time complexity t is omitted.

Let $\mathcal{A}$ be an adversary that returns a binary value; by $\mathcal{A}^{f(\cdot)}(X) = 1$ we refer to the event that $\mathcal{A}$ on input X and access to an oracle function $f(\cdot)$ returns 1. By *time complexity* of an algorithm we mean the running time, relative to some fixed model of computation plus the size of the description of the algorithm (program) using some fixed encoding method.

Definition 2.1 (Negligibility) *A function $f : \mathbb{N} \to \mathbb{R}$ is said to be* negligible *if*

$$\forall c \in \mathbb{R}^+, \exists \lambda_0 \in \mathbb{N} \text{ s.t. } \forall \lambda > \lambda_0 \text{ we have } f(\lambda) > \frac{1}{\lambda^c}\,.$$

Definition 2.2 (Negligible Probability) *We say that the probability of an event $E(\kappa)$ depending on a variable $\kappa \in \mathbb{N}$ is* negligible *if $Pr[E(\kappa)]$ is a negligible function.*

Definition 2.3 *Let $\{0,1\}^*$ be the set of binary strings and let $f : \{0,1\}^* \to \{0,1\}^*$. We call f a* one-way function *if*

1. *There exists an efficient algorithm that on input x returns $f(x)$*
2. *For any polynomial-time adversary $\mathcal{A}$, the following probability is negligible in κ:*

$$Pr[x \xleftarrow{\$} \{0,1\}^{\kappa}; y \leftarrow \mathcal{A}(1^{\kappa}, f(x)); f(y) = f(x)]\,.$$

Defining a Public-Key Encryption Scheme. An asymmetric-key encryption scheme consists of four PPT [19] algorithms (SETUP, KEYGEN, ENCRYPT, DECRYPT) and a corresponding key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. SETUP is a procedure for generating the parameters of the scheme, KEYGEN is an algorithm for generating keys, ENCRYPT is a procedure for encrypting, and DECRYPT is a procedure for decrypting. The functionalities of these algorithms is given in Figure 2.2.

Table 2.2: Algorithms of a public-key encryption scheme.

SETUP	Let κ be a security parameter. SETUP(1^{κ}) generates the global parameters of the scheme.
KEYGEN	KEYGEN takes the global parameters as input and generates the public encryption key pk and the corresponding private decryption key sk.
ENCRYPT	Let m be the plaintext. ENCRYPT [21] encrypts m into a ciphertext $c =$ENCRYPT(pk, m).
DECRYPT	DECRYPT decrypts c using sk. The output may be either the plaintext m or $\perp$ which is an invalidity symbol.

Computational Security

Shannon presented in [Sha48, Sha49] notions that form the mathematical foundation of modern cryptography. He defined the concept of *perfect secrecy*, introduced the idea of entropy of natural languages and statistical analysis. Moreover, he provided the first security proofs using probability theory and gave exact connections between provable security, key size, plaintext and ciphertext spaces.

The security of a cryptosystem relies on certain assumptions. Thus, we can distinguish between:
– Information-theoretically secure [22] cryptosystems: systems that no amount of computation can break

19. in the security parameter
21. Encryption may be deterministic or probabilistic.
22. Also called *unconditionally secure*.

– Computationally secure cryptosystems: based on the computational infeasibility of breaking them

In other words, no attack exists against unconditionally secure cryptosystems whereas attacks against computationally secure cryptosystems exist in theory but are infeasible in practice.

Definition 2.4 *The entropy (or,* equiv.*, uncertainty) $H(X)$ of a discrete random variable X with possible values x_i is defined as the expectation of the negative logarithm of the corresponding probability P*[23]*:*

$$H(X) = -\sum_i P(x_i) \log P(x_i) \,.$$

Definition 2.5 *A secret key cipher is perfect if and only if $H(M) = H(M|C)$, i.e., when the ciphertext C reveals no additional information about the message M.*

Corollary 2.1 *A perfect cipher is unconditionally (or information-theoretically) secure against a ciphertext only attack*[24] *(*COA*).*

Definition 2.6 (The Security of a Scheme) *Let t, ε be positive constants with $\varepsilon > 1$. We say that a scheme is (t, ε)-secure if every adversary $\mathcal{A}$ running for time at most t succeeds in breaking the scheme with probability at most ε.*

Definition 2.7 (The Asymptotic Security of a Scheme) *A scheme is secure if every probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with only negligible probability.*

We now turn our attention to proving the security of cryptosystems. We first give an intuition on how these proofs have to be constructed (having as building blocks reductionist proofs) and then discuss the models in which we can prove the security of a scheme.

Reductionist Proofs. A reductionist proof (be it uniform or non-uniform) establishes the security of a cryptographic scheme by showing how an adversary $\mathcal{A}$ that breaks the scheme can be used to solve an underlying problem which is assumed to be hard.

A problem X can be reduced to a problem Y if there is a constructive polynomial-time transformation that takes any instance of X and maps it to an instance of Y. Thus any algorithm for solving Y can be transformed into an algorithm solving X. Hence if X belongs to $\mathcal{P}$, and if X is reducible to Y, then $Y \in \mathcal{P}$.

The Adversarial Model. First, we formalise the notion of *security*: this definition is referred to as the *adversarial model* or the *security game*. More exactly, we model the attacker $\mathcal{A}$ as an interactive Turing machine, specify a challenger $\mathcal{C}$ and describe how $\mathcal{A}$ and $\mathcal{C}$ interact, including the winning condition for the attacker. There are security notion hierarchies for every type of scheme.

The Random Oracle Model (ROM). The Random Oracle Model (ROM). A random oracle is a black-box that receives as input binary strings (called *queries*) and returns randomly looking binary strings in response. The interaction with such an oracle is available to both honest parties and to adversaries. However the queries made by each party are assumed to be private to the querying party. The black-box has an important property called *consistence*: a repeated query will result in the *same* repeated response. In other words, a Random Oracle is, in a way, a "randomness made function".

Mixed Arguments. Some encryption schemes do not have direct reductions to hardness assumptions, hence, it is useful to divide the proof into smaller parts. This can be achieved by defining a sequence of *games*, starting from the initial security game, and slightly changing its rules until the game becomes impossible to win. The proof flows by showing that each game is hard to distinguish from the previous one, making the initial problem hard to distinguish from the final one. For showing that two successive games are hard to distinguish, one can resort to a hardness assumption or a statistical security argument.

23. assuming $P(x_i) \neq 0$
24. We assume that an attacker has access only to ciphertexts.

Security Notions for Symmetric-Key Cryptography

In the following definitions, LOR will denote left or right indistinguishability, ROR - real or random indistinguishability, FTG - find then guess indistinguishability and SEM will denote semantic security. LOR and ROR were initially defined by Bellare *et al.* [BDJR97], whereas FTG and SEM are symmetric-key setting adaptations of Goldwasser and Micali's definitions [GM84]. FTG and SEM will not be detailed or formally defined next, as our original results are mostly public-key related.

In the definitions below, $\mathcal{K}$ will be the key generation algorithm and $k \in \mathbb{N}$ a security parameter. K will denote a key and Exp will be an *experiment*. In the following Definitions (2.8 to 2.11) $\mathcal{SKE}$ will be a symmetric-key encryption scheme. We let $b \in \{0,1\}$.

We follow the description and definitions in [BDJR97].

LOR $\mathcal{A}$ is allowed queries of the form (x_0, x_1) where x_0, x_1 are equal-length messages. We define the LOR oracle $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, where $b \in \{0,1\}$, to take the input (x_0, x_1) and: if $b = 0$ it computes $C \leftarrow \mathcal{E}_K(x_0)$ and returns C; else it computes $C \leftarrow \mathcal{E}_K(x_1)$ and returns C. To model CCAs we allow the adversary to also have access to a decryption oracle $\mathcal{D}_K(\cdot)$.

Definition 2.8 (LOR-CPA **Security Game and Advantage)** *Let $\mathcal{A}_{\text{CPA}}$ be an adversary that has access to the oracle $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$. We consider the following game:*

1. $\text{Exp}^{\text{LOR-CPA}-b}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k)$
2. $K \xleftarrow{\$} \mathcal{K}(k)$
3. $d \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))}_{\text{CPA}}(k)$
4. *Return* d

We define the advantage of the adversary as:

$$\text{Adv}^{\text{LOR-CPA}}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k) = Pr[\text{Exp}^{\text{LOR-CPA}-1}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k) = 1] - Pr[\text{Exp}^{\text{LOR-CPA}-0}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k) = 0]\,.$$

Definition 2.9 (LOR-CCA **Security Game and Advantage)** *Let $\mathcal{A}_{\text{CCA}}$ be an adversary that has access to the oracles $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ and $\mathcal{D}_K(\cdot)$. We consider the following game:*

1. $\text{Exp}^{\text{LOR-CCA}-b}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k)$
2. $K \xleftarrow{\$} \mathcal{K}(k)$
3. $d \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}_{\text{CCA}}(k)$
4. *Return* d

We define the advantage of the adversary as:

$$\text{Adv}^{\text{LOR-CCA}}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k) = Pr[\text{Exp}^{\text{LOR-CCA}-1}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k) = 1] - Pr[\text{Exp}^{\text{LOR-CCA}-0}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k) = 0]\,.$$

ROR The idea is that $\mathcal{A}$ cannot distinguish the encryption of a text from an encryption of an equal-length meaningless string. We define the ROR oracle $\mathcal{E}_K(\mathcal{RR}(\cdot,b))$ where $b \in \{0,1\}$, to take the input x and: if $b = 0$ it computes $C \leftarrow \mathcal{E}_K(x)$ and returns C; else it computes $C \leftarrow \mathcal{E}_K(r)$, where $r \xleftarrow{\$} \{0,1\}^{|x|}$ and returns C.

Definition 2.10 (ROR-CPA**)** *Let $\mathcal{A}_{\text{CPA}}$ be an adversary that has access to the oracles $\mathcal{E}_K(\mathcal{RR}(\cdot,\cdot,b))$ and $\mathcal{D}_K(\cdot)$. We consider the following game:*

1. $\text{Exp}^{\text{ROR-CPA}-b}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k)$
2. $K \xleftarrow{\$} \mathcal{K}(k)$
3. $d \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{RR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}_{\text{CPA}}(k)$
4. *Return* d

We define the advantage of the adversary as:

$$\text{Adv}^{\text{ROR-CPA}}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k) = Pr[\text{Exp}^{\text{ROR-CPA}-1}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k) = 1] - Pr[\text{Exp}^{\text{ROR-CPA}-0}_{\mathcal{SKE},\mathcal{A}_{\text{CPA}}}(k) = 0]\,.$$

Definition 2.11 (ROR-CCA**)** *Let $\mathcal{A}_{\text{CCA}}$ be an adversary that has access to the oracles $\mathcal{E}_K(\mathcal{RR}(\cdot,\cdot,b))$ and $\mathcal{D}_K(\cdot)$. We consider the following game:*

1. $\text{Exp}^{\text{ROR-CCA}-b}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k)$
2. $K \xleftarrow{\$} \mathcal{K}(k)$
3. $d \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{RR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}_{\text{CCA}}(k)$
4. *Return* d

We define the advantage of the adversary as:

$$\text{Adv}^{\text{ROR-CCA}}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k) = Pr[\text{Exp}^{\text{ROR-CCA}-1}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k) = 1] - Pr[\text{Exp}^{\text{ROR-CCA}-0}_{\mathcal{SKE},\mathcal{A}_{\text{CCA}}}(k) = 0]\,.$$

[BDJR97] lists relations between these different notions. Mainly, ROR $\Leftrightarrow$ LOR, LOR $\Rightarrow$ FTG and FTG $\Leftrightarrow$ SEM under any type of attack.

Complexity Theory

Decision Problems. A *decision problem* is a problem in a formal system whose solution has to be a yes or a no. Complexity theory's main goal is to understand and quantify the difficulty of solving specific decision problems. A set of problems of related complexity is referred to as a *complexity class*.

The Turing machine, introduced by Alan Turing in 1936 [Tur36] is the standard computational model on which the decision problems' theory is based.

A Turing machine consists of a finite program attached to a reading or writing head moving on an infinite tape. The tape is divided into squares, each capable of storing one symbol from a finite alphabet `Alph`* which includes a blank symbol `blank`. Each machine has a specified input alphabet `Alph`, which is a subset of `Alph`*, without `blank`. At a given point during a computation the machine is in a state q which is in a specified finite set Q of possible states. At first, a finite input string over `Alph` is written on adjacent squares of the tape, all other squares are blank (contain `blank`), the head scans the left-most symbol of the input string, and the machine is in the initial state q_0. At each step, the machine is in a state q and the head is scanning a tape square containing a tape symbol s. The action performed depends on the pair (q, s) and is specified by the machine's transition function τ. The action consists of printing a symbol on the scanned square, moving the head left or right one square, and assuming a new state.

We further give the definitions of both the $\mathcal{P}$ and $\mathcal{NP}$ problem classes.

Definition 2.12 (Complexity Class $\mathcal{P}$) *A decision problem belongs to the $\mathcal{P}$ class if there exists a polynomial-time algorithm able to solve it, i.e. given an input of length n, there exists an algorithm that produces the answer in a number of steps polynomial in n.*

The decision problems in $\mathcal{P}$ are the decision problems that can be solved by a deterministic Turing machine in polynomial time.

Definition 2.13 (Complexity Class $\mathcal{NP}$) *A decision problem belongs to the class $\mathcal{NP}$ if a yes instance*[25] *of the problem can be verified in polynomial time.*

The $\mathcal{P}$ *versus* $\mathcal{NP}$ Problem. The $\mathcal{P}$ *versus* $\mathcal{NP}$ problem is central in complexity theory. This problem can be simply stated as follows: "*is* $\mathcal{P} = \mathcal{NP}$?". This is a major open problem, listed on Clay Mathematics Institute's millennium problems list.

Hardness Assumptions

A necessary condition for cryptosystem security is the assumption that some underlying problem is hard to solve. These concepts must be well established.

Hardness assumptions are the cornerstone of provable security for public-key encryption, a concept which has become increasingly important over the years. Provable security is also one of the prime

25. An instance for which the purported answer is yes.

expertise areas of the ENS Cryptography Group. Thus, hardness assumptions are among the core ingredients required for the development of public key cryptosystems. A short description of the most common hardness assumptions is given next.

Discrete Logarithm Problem instantiations play an important role in this thesis, *cf.* Section 4.1.

1. **The Discrete Logarithm Problem -** DLP

   Definition 2.14 (The Discrete Logarithm Problem - DLP**)** *Let G be a subgroup of $\mathbb{Z}_p^*$. Let g be a generator of G (of order q). Given $g, p, h \in_R G$, find a such that $h = g^a$.*
   The number a is called the discrete logarithm of h to the base g and is denoted by $\log_g h$.

   Definition 2.15 (The Computational Diffie-Hellman Problem - CDHP**)** *Given a finite cyclic group G of order q, a generator g of G, and two elements g^a and g^b, find the element g^{ab}.*

   Definition 2.16 (The Decisional Diffie-Hellman Problem - DDHP**)** *Given a finite cyclic group G, a generator g of G, three elements g^a, g^b and g^c, decide whether the elements g^c and g^{ab} are equal.*

2. DLP **versions for** ECC
   The elliptic curve variant of Definition 2.14 is given below. For a very short introduction to the terminology used in elliptic curve cryptography, the reader may consult Section 3.4.

   Definition 2.17 (The Elliptic Curve Discrete Logarithm Problem - ECDLP**)** *Let E be an elliptic curve over the finite field $\mathbb{F}_p$ and let P and Q be points in $E(\mathbb{F}_p)$. The elliptic curve discrete logarithm problem consists in finding $n \in \mathbb{N}$ such that $Q = [n]P$.*

   For symmetry with the DLP for G, we denote this integer n by $n = \log_P Q$ and we call n the *elliptic discrete logarithm of Q with respect to P*.
   Note. There may be settings in which $\log_P Q$ is undefined, but in practical cryptographic applications $\log_P Q$ exists.

   Pairing-Based Cryptography. Let g be a generator for a group G of prime order q, and let e be a bilinear map on G.

   Definition 2.18 *Let G_1, G_2 be two additive cyclic groups of prime order p, and G_3 a multiplicative cyclic group of order p. A pairing is an efficiently computable map $e : G_1 \times G_2 \to G_3$ satisfying*
   – **bilinearity***: $\forall a, b \in F_p^*$ and $\forall P \in G_1, Q \in G_2$ we have $e([a]P, [b]Q) = e(P, Q)^{ab}$*
   – **non-degeneracy***: $e(P, Q) \neq 1$*

   Definition 2.19 (The Bilinear Diffie-Hellman Problem - BDHP**)** *Let G, G_T be two cyclic groups of a large prime order p. Let $e \in G_T$ be a bilinear pairing. Given $g, g^x, g^y, g^z \in G^4$, compute $e(g, g)^{xyz} \in G_T$.*

   Definition 2.20 (The Bilinear Decisional Diffie-Hellman Problem - BDDHP**)** *Let G, G_T be two cyclic groups of a large prime order p. Let $e \in G_T$ be a bilinear pairing. Given $g, g^x, g^y, g^z \in G$ and $e(g, g)^w \in G_T$, decide if $w = xyz \bmod p$.*

   Obviously, BDHP $\Rightarrow$ BDDHP: Indeed if we can compute the bilinear pairing $e(g, g)^{xyz}$, then we solve BDDHP by comparing to the provided value $e(g, g)^w$.

3. **Factorization-Related Problems -** FACT **and** ERP

   Definition 2.21 *Given a positive integer n, find its prime factors, i.e., find the pairwise distinct primes p_i and positive integer powers e_i such that $n = p_1^{e_1} ... p_n^{e_n}$.*

   Definition 2.22 (The e-th Root Problem - ERP**)** *Given a group G of unknown order, a positive integer $e < |G|$ and an element $a \in G$, find an element $b \in G$ such that $b^e = a$.*

   Note. RSA relies on the difficulty of solving equations of the form $x^e = c \bmod N$, where e, c, and N are known and x is the unknown. In other words, the security of RSA relies on the assumption that it is difficult to compute e-th roots modulo N, *i.e.* the ERP in $\mathbb{Z}_N$.
   Proving or refuting the equivalence between RSA and FACT is a major open problem in cryptography, about which there are only partial and preliminary results so far.

4. **Residuosity Problems -** QRP **and** HRP

Definition 2.23 *Let $a, n, m \in \mathbb{N}$ with $\gcd(a, n) = 1$. a is called an m-th residue $\bmod n$ if there exists an integer x such that $a \equiv x^m \bmod n$.*

The residuosity problem may refer to *quadratic* or to *higher* residues.

Definition 2.24 *Let n be the product of two primes p and q. An element $a \in \mathbb{Z}_n$ is a quadratic residue modulo n (or a square) if there exists $w \in \mathbb{Z}_n$ such that $w^2 \equiv a \bmod n$. If there exist no such $w \in \mathbb{Z}_n$, a is called a quadratic non-residue.*

Definition 2.25 (The Quadratic Residuosity Problem - QRP) *Given $a, n \in \mathbb{N}$, $0 \leq a < n$, decide if a is a quadratic residue.*

Definition 2.26 (The Higher Residuosity Problem - HRP) *Given $a, n, m \in \mathbb{N}$, $0 \leq a < n$, $\gcd(a, n) = 1$ decide if a is an m-th residue.*

Note. QRP's intractability is the basis for the security of Goldwasser–Micali's cryptosystem [GM82], the first provably secure probabilistic public key encryption scheme[26]. Paillier's cryptosystem [Pai99] is the best known example of a scheme whose underlying hardness assumption is the HRP.

Security Notions for Public-Key Cryptography

In the following definitions, IND will denote indistinguishability and NM will denote non-maleability. IND was initially defined by Goldwasser and Micali [GM84], and NM by Dolev, Dwork and Naor [DDN00]. The NM notion will not be detailed or formally defined in this manuscript. We refer the reader to [BDJR97] for a precise description of this concept. For defining IND we follow the description given in [BDPR98].

Experiments. Let A be a probabilistic algorithm and denote by $A(x_1, x_2, \ldots; r)$ is the result of running A on inputs $x_1, x_2, \ldots$ and coins r.

We let $y \leftarrow A(x_1, x_2, \ldots)$ denote the experiment of picking r at random and let $y = A(x_1, x_2, \ldots; r)$.

We say that y is a potential output of $A(x_1, x_2, \ldots)$ if $\exists r$ such that $A(x_1, x_2, \ldots; r) = y$.

IND. A public key encryption scheme $\mathcal{PKE}$ satisfies the property IND if the distributions A_{m_1} and A_{m_2} are *computationally indistinguishable*[27] for all $m_1, m_2 \in \mathcal{M}$ such that $|m_1| = |m_2|$ where

$$A_{m_i} = \left\{pk, \mathcal{PKE}.\,\textsc{Encrypt}(pk, m_i) : (pk, sk) \xleftarrow{\$} \mathcal{PKE}.\textsc{KeyGen}(\lambda)\right\}, \text{ for } i \in \{1, 2\}\,.$$

The commonly desired security properties of public-key encryption are *indistinguishability under chosen plaintext attack* (IND-CPA) or semantic security, *indistinguishability under chosen ciphertext attack* (IND-CCA1) and *indistinguishability under adaptive ciphertext attack* (IND-CCA2) defined by the security games of Definitions 2.27, 2.29 and 2.31. Weaker security notions whose presentations we omit here are *one-wayness under chosen plaintext attack* and *under chosen ciphertext attack*, (OW-CPA and OW-CCA). We refer the reader to [Poi05] for a detail description of OW-CPA and OW-CCA.

Definition 2.27 (IND-CPA Security Game and Advantage) *The IND-CPA security game $G^{\text{IND-CPA}}_{\mathcal{PKE}}$ is defined as a protocol between the challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{C}$ runs $(sk, pk) \leftarrow \mathcal{PKE}.\textsc{KeyGen}(\lambda)$ and sends pk to $\mathcal{A}$*
2. *$\mathcal{A}$ chooses two messages m_0 and m_1 and sends them to $\mathcal{C}$*
3. *$\mathcal{C}$ chooses a uniform random bit b and encrypts one of the two message accordingly:*

$$c \leftarrow \mathcal{PKE}.\,\textsc{Encrypt}(pk, m_b)$$

4. *$\mathcal{A}$ sends a guess b' to $\mathcal{C}$*

26. In the case of a probabilistic encryption scheme a message is encrypted into one of many possible ciphertexts.
27. Two probabilities are *computationally indistinguishable* if no efficient algorithm can make the difference between them.

5. *$\mathcal{C}$ outputs 1 if the guess was correct, that is if $b = b'$, 0 otherwise*

The advantage of an IND-CPA *adversary $\mathcal{A}$ against this game is defined as:*

$$\mathrm{Adv}^{\text{IND-CPA}}_{\mathcal{PKE}}(\mathcal{A}) = \Pr\left[G^{\text{IND-CPA}}_{\mathcal{PKE}}(\mathcal{A}) = 1\right] - 1/2$$

Definition 2.28 (IND-CPA **Security)** *A public-key encryption scheme $\mathcal{PKE}$ is said to be* IND-CPA *secure if for any adversary $\mathcal{A}$ that runs in probabilistic polynomial time (PPT) in the security parameter λ, $\mathcal{A}$'s advantage $\mathrm{Adv}^{\text{IND-CPA}}_{\mathcal{PKE}}(\mathcal{A})$ is negligible in λ.*

Definition 2.29 (IND-CCA1 **Security Game and Advantage)** *The* IND-CCA1 *security game $G^{\text{IND-CCA1}}_{\mathcal{PKE}}$ is defined as a protocol between the challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{C}$ runs $(sk, pk) \leftarrow \mathcal{PKE}.\textsc{KeyGen}(\lambda)$ and sends pk to $\mathcal{A}$*
2. *$\mathcal{A}$ may perform polynomially many encryptions, calls to the decryption oracle based on arbitrary ciphertexts, or other operations*
3. *Eventually, $\mathcal{A}$ submits two distinct chosen plaintexts m_0 and m_1 to $\mathcal{C}$.*
4. *$\mathcal{C}$ chooses a random bit b and encrypts one of the two message accordingly:*

$$c \leftarrow \mathcal{PKE}.\textsc{Encrypt}(pk, m_b)$$

5. *$\mathcal{A}$ may* **not** [28] *make further calls to the decryption oracle*
6. *$\mathcal{A}$ sends a guess b' to $\mathcal{C}$*
7. *$\mathcal{C}$ outputs 1 if the guess was correct, that is if $b = b'$, 0 otherwise*

The advantage of an IND-CCA1 *adversary $\mathcal{A}$ against this game is defined as:*

$$\mathrm{Adv}^{\text{IND-CCA1}}_{\mathcal{PKE}}(\mathcal{A}) = \Pr\left[G^{\text{IND-CCA1}}_{\mathcal{PKE}}(\mathcal{A}) = 1\right] - 1/2$$

Definition 2.30 (IND-CCA1 **Security)** *A public-key encryption scheme $\mathcal{PKE}$ is said to be* IND-CCA1 *secure if for any adversary $\mathcal{A}$ that runs in probabilistic polynomial time (PPT) in the security parameter λ, $\mathcal{A}$'s advantage $\mathrm{Adv}^{\text{IND-CCA1}}_{\mathcal{PKE}}(\mathcal{A})$ is negligible in λ.*

Definition 2.31 (IND-CCA2 **Security Game and Advantage)** *The* IND-CCA2 *security game $G^{\text{IND-CCA2}}_{\mathcal{PKE}}$ is defined as a protocol between the challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{C}$ runs $(sk, pk) \leftarrow \mathcal{PKE}.\textsc{KeyGen}(\lambda)$ and sends pk to $\mathcal{A}$*
2. *$\mathcal{A}$ may perform polynomially many encryptions, calls to the decryption oracle based on arbitrary ciphertexts, or other operations*
3. *Eventually, $\mathcal{A}$ submits two distinct chosen plaintexts m_0 and m_1 to $\mathcal{C}$*
4. *$\mathcal{C}$ chooses a uniform random bit b and encrypts one of the two message accordingly:*

$$c \leftarrow \mathcal{PKE}.\textsc{Encrypt}(pk, m_b)$$

5. *$\mathcal{A}$ may make further calls to the encryption or decryption oracles, but may not submit the challenge ciphertext c to $\mathcal{C}$*
6. *$\mathcal{A}$ sends a guess b' to $\mathcal{C}$*
7. *$\mathcal{C}$ outputs 1 if the guess was correct, that is if $b = b'$, 0 otherwise*

The advantage of an IND-CCA2 *adversary $\mathcal{A}$ against this game is defined as:*

$$\mathrm{Adv}^{\text{IND-CCA2}}_{\mathcal{PKE}}(\mathcal{A}) = \Pr\left[G^{\text{IND-CCA2}}_{\mathcal{PKE}}(\mathcal{A}) = 1\right] - 1/2$$

Definition 2.32 (IND-CCA2 **Security)** *A public-key encryption scheme $\mathcal{PKE}$ is said to be* IND-CCA2 *secure if for any adversary $\mathcal{A}$ that runs in probabilistic polynomial time (PPT) in the security parameter λ, its advantage $\mathrm{Adv}^{\text{IND-CCA2}}_{\mathcal{PKE}}(\mathcal{A})$ is negligible in λ.*

The implication relations between the above security notions are given in Figure 2.10 and [BDPR98].

28. Step *5* stresses the difference between IND-CCA1 and IND-CCA2. Thus, we underline that for IND-CCA1, $\mathcal{A}$ will not be allowed to interact with the decryption oracle after step *4*.

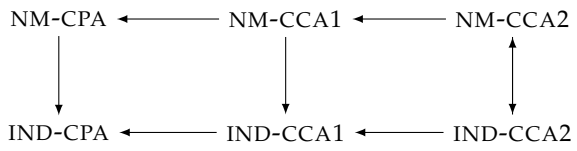


Figure 2.10: Relations between public-key security notions.

Post-Quantum Cryptography

The genesis of quantum computation dates back to 1981, when Nobel prize winner Feynman [Fey82] came up with the idea of using quantum mechanical effects to build a computer that would have outstanding performance in terms of computational speed. It was only in 1993 that Bernstein and Vazirani [BV93] understood the importance of Feynman's remarkable ideas and in 1994 that Shor [Sho97] pointed out that the advent of quantum computation will render factoring and discrete logarithm computation efficient.

The above naturally gave birth to the field of *post-quantum cryptography* [Ber09]. Recently, the National Institute for Standards and Technology (NIST) has shown a growing interest in the standardization of quantum resistant cryptography and sponsored several workshops on this topic. Moreover, two important post-quantum cryptography related projects are funded by the European Union: PQCrypto [pqc] and SAFEcrypto [saf].

During the last years, one of the main goals in this area was increasing the efficiency of the already established schemes.

The following are post-quantum cryptography algorithm (or underlying hard problem) families:

Lattice-Based. A *lattice* is a set of points in the n-dimensional space, having a periodic structure defined as follows:

Definition 2.33 *Let* $\mathbf{b_1}, ..., \mathbf{b_n} \in \mathbb{R}^n$ *be* n*-linearly independent vectors* [29]*. The lattice* L *generated by* $\mathbf{b_1}, ..., \mathbf{b_n}$ *is the set of vectors*

$$L(\mathbf{b_1}, ..., \mathbf{b_n}) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b_i} | x_i \in \mathbb{Z} \right\}.$$

The underlying security assumptions of lattice-based cryptosystems imply the hardness of basis reduction and related problems in random lattices [Reg06]. Based on Ajtai's seminal work [Ajt96] [30], Ajtai and Dwork [AD97] presented a cryptosystem shown to be provably secure under the assumption that a certain lattice problem is difficult in the worst-case. However, Nguyen and Stern have shown that the Ajtai-Dwork cryptosystem is only of theoretical interest [NS98] [31]. Another asymmetric lattice-based key encryption system that caught cryptographers' attention is the NTRU (Hoffstein–Pipher–Silverman) cryptosystem [HPS98].

The attempts to solve lattice problems by quantum algorithms have had little success (or even not at all). The periodicity finding technique [NC11], used by Shor's quantum factoring and other related algorithms [Sim97, Hal02], is not applicable to lattice problems. Periodicity finding is mainly based on quantum computers' capability of being in many states at the same time: to compute a function's period the device evaluates the function at all points simultaneously.

It is conjectured that no polynomial-time quantum algorithm can approximate lattice problems to within polynomial factors [MR09].

29. The vectors $\mathbf{b_1}, ..., \mathbf{b_n}$ are called the *basis* of the lattice.
30. Ajtai [Ajt96] found the first connection between the worst-case and the average-case complexity of the shortest vector problem (SVP) [MR09].
31. Nguyen and Stern [NS98] conducted an heuristic attack to recover the private key, given only the public key.

Error Correction Code-Based. The first error-correction based cryptosystem was proposed by McEliece [McE78] in 1978. Its security is based on the hardness of decoding random linear error-correcting codes. Finding a codeword of given weight in a linear binary code is an $\mathcal{NP}$-complete problem [BMvT06].

The initial parameters of the McEliece cryptosystem were broken by Bernstein *et al.* [BLP08]. However, new parameters secure against all known attacks were proposed in [BLP08].

Hash-Based. In 1979 Merkle proposed a hash-based signature scheme [Mer79]. While being quantum-resistant, Merkle's scheme is stateful. This means that the signer must keep a state (history record) that evolves as messages are being signed.

At EUROCRYPT 2015, Bernstein *et al.* [BHH+15] presented stateless hash-based signatures.

Multivariate Quadratic Equations (MQE)-Based. MQE-based cryptosystems [FD86, MI88] base their security on the conjectured hardness of solving some polynomial equations systems. The exact hardness of breaking MQE-based cryptosystems is still not thoroughly understood.

Note: The impact of quantum computing on symmetric algorithms is usually smaller. Considering as a typical example the AES [DR02], even if Grover's algorithm [Gro96] can be used to speed up brute-force attacks against symmetric ciphers and collision-resistant hash functions, it was shown [BBBV97] that an ℓ-bit key provides $\frac{\ell}{2}$ bits of quantum security. Hence, in a quantum-computer world, AES-256 would still offer 128 bits of security.

2.2 Thesis Outline

A summary of the contributions included in this thesis is given in Section 2.3, as well as a full publication list and our submission to the *Competition for Authenticated Encryption: Security, Applicability, and Robustness* (CAESAR).

Chapter 3 presents the underlying theoretical foundations related to our results.

Chapter 3 provides a short introduction to hash functions and message authentication codes (MACs). Chapter 3 also presents authenticated encryption (both the generic composition paradigm [BN08] [32] and dedicated solutions [33] [Jut01, GD01, RBBK01]) and overviews digital signature schemes. Elliptic curves are introduced and their role in cryptography is discussed, in view of the cryptanalysis results presented in Section 5.6.

Our original contributions are structured in two chapters: Protocol Design (Chapter 4) and Algorithms for Embedded Cryptography (Chapter 5).

Chapter 4 presents a provably secure co-signature protocol and a provably secure authenticated encryption scheme. Chapter 4 also includes a lightweight multiparty Fiat-Shamir authentication protocol.

Chapter 5 includes all the contributions listed in Section 2.2 as well as a fault attack on ECC implementations, building upon [NSS04].

Therefore, the manuscript's core results revolve around: protocol design, algorithmic improvements and attacks.

Protocol Design. Designing secure and efficient cryptographic protocols is challenging, as protocol-level security proofs are much more complex than proofs of cryptographic primitives.

When discussing multiparty protocols, a very interesting design approach, called *rational protocol design*, connects cryptography and game theory. In traditional security definitions, threats are modelled by an *adversary*. *Rational cryptography* doesn't assume honest and corrupted parties, but rather rational parties, motivated by certain utility functions. In other words, rational cryptography takes into consideration the *incentives that lead parties to deviate from their prescribed behavior* [GKM+13]. In this thesis, we focus on the traditional protocol design security perspective and not on *rational cryptography*.

The main result of the thesis, presented in Section 4.1, is a novel form of fairness, called *legal fairness*, that does not rely on third parties in the context of contract signing.

A second constructive result is given in Section 4.3: a mode of operation for applying a compression function to build a nonce-based authenticated encryption with associated data (AD). Associated data can be seen as a part of the message (for example, some header information) which needs integrity but not confidentiality protection. Thus, AD is authenticated but not encrypted.

Section 4.2 presents a distributed Fiat-Shamir authentication protocol enabling network node authentication using very few communication rounds.

Algorithmic Improvements. Besides being secure, real world cryptographic applications must also be efficient.

Thus, most of the results presented in Chapter 5 concern algorithmic speed-ups. We describe these improvements from a lightweight cryptography perspective. *Lightweight cryptography* is cryptography suitable for constrained devices in which designers must strike trade-offs between performance, security, and cost. The constraints may be computing power, memory, bandwidth, or security.

Section 5.2 includes a new method allowing to double the speed of Barrett's algorithm by using specific composite moduli. Section 5.4 describes a new multiplication algorithm, especially suited for 8-bit lightweight microprocessors when one of the operands is a known constant. Section 5.3.1 describes new BCH speed-up strategies using polynomial versions of Barrett's algorithm. Section 5.3.2 describes a new error-correcting code (ECC) inspired by the Naccache-Stern cryptosystem [NS97, CMNS08] which

32. The generic composition paradigm refers to combining an encryption scheme and a MAC.
33. A dedicated solution refers to the providing of both privacy and authenticity in a single scheme.

happens to be more efficient than some established ECCs for certain sets of parameters. Section 5.5 presents a new method to regulate the pace of random bits outputted by the von Neumann randomness extractor.

Fault Attacks. Fault attacks represent a major threat for cryptographic applications. Some of the most common side channel attacks are power analysis (Simple Power Analysis and Differential Power Analysis), electromagnetic radiation analysis and fault injection attacks.

After an introduction to fault injection attacks at the beginning of Chapter 5, we propose a new fault attack on ECC implementations. The attack consists in injecting a fault during the projective-to-affine conversion process so the erroneous results reveals information about Z (third coordinate of a point in the Jacobian projective coordinates system). Several faulty results permit to recover Z and hence, expose the target to an attack described in [NSS04].

2.3 Publications

The main results of this thesis are presented in the current section: published papers, pre-prints and the submission to the cryptographic competition CAESAR.

Legally Fair Contract Signing without Keystones [To Appear]

with Houda Ferradi, Rémi Géraud, David Naccache and David Pointcheval

Abstract. In two-party computation, achieving both fairness and guaranteed output delivery is well known to be impossible. Despite this limitation, many approaches provide solutions of practical interest by weakening somewhat the fairness requirement. Such approaches fall roughly in three categories: "gradual release" schemes assume that the aggrieved party can eventually reconstruct the missing information; "optimistic schemes" assume a trusted third party arbitrator that can restore fairness in case of litigation; and "concurrent" or "legally fair" schemes in which a breach of fairness is compensated by the aggrieved party having a digitally signed cheque from the other party (called the keystone).

In this paper we describe and analyse a new contract signing paradigm that doesn't require keystones to achieve legal fairness, and give a concrete construction based on Schnorr signatures which is compatible with standard Schnorr signatures and provably secure.

Note. The details of this paper are given in Chapter 4. My contribution to this paper consisted of developing the idea.

Offset Merkle-Damgård (OMD) v.1: A CAESAR Proposal

OMD: A Compression Function Mode of Operation for Authenticated Encryption [CMN+14, Ber]

with Simon Cogliani, David Naccache, Rodrigo Portella, Reza Reyhanitabar, Serge Vaudenay and Damian Vizár

Abstract. We propose the Offset Merkle-Damgard (OMD) scheme, a mode of operation to use a compression function for building a nonce-based authenticated encryption with associated data. In OMD, the parts responsible for privacy and authenticity are tightly coupled to minimize the total number of compression function calls: for processing a message of ℓ blocks and associated data of a blocks, OMD needs $\ell + a + 2$ calls to the compression function (plus a single call during the whole lifetime of the key). OMD is provably secure based on the standard pseudorandom function (PRF) property of the compression function. Instantiations of OMD using the compression functions of SHA-256 and SHA-512, called OMD-SHA256 and OMD-SHA512, respectively, provide much higher quantitative level of security

compared to the AES-based schemes. OMD-SHA256 can benefit from the new Intel SHA Extensions on next-generation processors.

Note. Published in the proceedings of *SAC 2014*. The results mentioned above are presented in detail in Chapter 4. The compression functions of SHA-256 and SHA-512 are recalled in Appendix A.

Winning he CAESAR competition was amongst the motivation for developing OMD. We were recently pleased to learn that our scheme was listed as a *second round candidate*. My contribution to this paper consisted of developing the idea.

Fault Attacks on Projective-to-Affine Coordinates Conversion [MMNT13]

with Cédric Murdica, David Naccache and Mehdi Tibouchi

Abstract. This paper presents a new type of fault attacks on elliptic curves cryptosystems.

At EUROCRYPT 2004, Naccache *et al.* showed that when the result of an elliptic curve scalar multiplication $[k]P$ (computed using a fixed scalar multiplication algorithm, such as double-and-add) is given in projective coordinates, an attacker can recover information on k. The attack is somewhat theoretical, because elliptic curve cryptosystems implementations usually convert scalar multiplication's result back to affine coordinates before outputting $[k]P$.

This paper explains how injecting faults in the final projective-to-affine coordinate conversion enables an attacker to retrieve the projective coordinates of $[k]P$, making Naccache *et al.*'s attack also applicable to implementations that output points in affine coordinates. As a result, such faults allow the recovery of information about k.

Note. Published in the proceedings of *COSADE 2013*. A detailed version of this work is presented in Chapter 5. My contribution to this paper consisted of developing the idea.

Double-Speed Barrett Moduli [GMN15c, GMN15a]

with Rémi Géraud and David Naccache

Abstract. Modular multiplication and modular reduction are the atomic constituents of most public-key cryptosystems. Amongst the numerous algorithms for performing these operations, a particularly elegant method was proposed by Barrett. This method builds the operation $a \bmod b$ from bit shifts, multiplications and additions in $\mathbb{Z}$. This allows to build modular reduction at very marginal code or silicon costs by leveraging existing hardware or software multipliers.

This paper presents a method allowing to double the speed of Barrett's algorithm by using specific composite moduli. This is particularly useful for lightweight devices where such an optimization can make a difference in terms of power consumption, cost and processing time. The generation of composite moduli with a predetermined portion is a well-known technique and the use of such moduli is considered, *in statu scientiæ*, as safe as using randomly generated composite moduli.

Note. Published in *Kahn Festschrift LNCS volume 9100, 2015*. This paper is described in Chapter 5. My contribution to this paper as a leading author consisted of developing the idea and implementing it in software.

Public-Key Based Lightweight Swarm Authentication [To appear]

with Simon Cogliani, Rémi Géraud, Rodrigo Portella do Canto and David Naccache

Abstract. We describe a lightweight algorithm performing whole-network authentication in a distributed way. This protocol is more efficient than one-to-one node authentication: it results in less communication, less computation, and overall lower energy consumption.

The security of the proposed algorithm can be reduced to the RSA hardness assumption, and it achieves zero-knowledge authentication of a network in a time logarithmic in the number of nodes.

Note. A detailed version of this paper is presented in Chapter 5. As a leading author I worked on developing the idea.

Applying Cryptographic Acceleration Techniques to Error Correction [MNPdCS15, GMN+15b]

with Rémi Géraud, Rodrigo Portella do Canto, David Naccache and Emil Simion

Abstract. Modular reduction is the basic building block of many public-key cryptosystems. BCH codes require repeated polynomial reductions modulo the same constant polynomial. This is conceptually very similar to the implementation of public-key cryptography where repeated modular reduction in $\mathbb{Z}_n$ or $\mathbb{Z}_p$ are required for some fixed n or p. It is hence natural to try and transfer the modular reduction expertise developed by cryptographers during the past decades to obtain new BCH speed-up strategies. Error correction codes (ECCs) are deployed in digital communication systems to enforce transmission accuracy. BCH codes are a particularly popular ECC family. This paper generalizes Barrett's modular reduction to polynomials to speed up BCH ECCs. A BCH$(15, 7, 2)$ encoder was implemented in Verilog and synthesized. Results show substantial improvements when compared to traditional polynomial reduction implementations. We present two BCH code implementations (regular and pipelined) using Barrett polynomial reduction. These implementations, are respectively 4.3 and 6.7 faster than an improved BCH LFSR design. The regular Barrett design consumes around 53% less power than the BCH LFSR design, while the faster pipelined version consumes 2.3 times more power than the BCH LFSR design.

Note. Published in the proceedings of *SECITC 2015*. This paper is described in Chapter 5. My contribution to this paper as a leading author consisted of developing the idea and implementing it in software.

A Number-Theoretic Error-Correcting Code [BCG+15b, BCG+15a]

with Eric Brier, Jean-Sébastien Coron, Rémi Géraud and David Naccache

Abstract. In this paper we describe a new error-correcting code (ECC) inspired by the Naccache-Stern cryptosystem. While by far less efficient than Turbo codes, the proposed ECC happens to be more efficient than some established ECCs for certain sets of parameters.

The new ECC adds an appendix to the message. The appendix is the modular product of small primes representing the message bits. The receiver recomputes the product and detects transmission errors using modular division and lattice reduction.

Note: Published in the proceedings of *SECITC 2015*. This paper is presented in Chapter 5. My contribution to this paper consisted of developing the idea.

Backtracking-Assisted Multiplication [FGM+15b]

with Houda Ferradi, Rémi Géraud, David Naccache and Hang Zhou

Abstract. This paper describes a new multiplication algorithm, particularly suited to lightweight microprocessors when one of the operands is known in advance. The method uses backtracking to find a multiplication-friendly encoding of one of the operands.

A 68HC05 microprocessor implementation shows that the new algorithm indeed yields a twofold speed improvement over classical multiplication for 128-byte numbers.

Note: This paper is described in Chapter 5. My contribution to this paper consisted of developing the idea and implementing it in software.

Regulating the Pace of von Neumann Extractors [FGM+15a]

with Houda Ferradi, Rémi Géraud, David Naccache and Amaury de Wargny

Abstract. In a celebrated paper published in 1951, von Neumann presented a simple procedure allowing to correct the bias of random sources. This device outputs bits at irregular intervals. However, cryptographic hardware is usually synchronous.

This paper proposes a new building block called Pace Regulator, inserted between the randomness consumer and the von Neumann extractor to streamline the pace of random bits.

Note: This paper is described in Chapter 5. My contribution to this paper consisted of developing the idea.

Lightweight Cryptography for RFID Tags [MO12]

with Khaled Ouafi

Abstract. Combining the terms "cryptography" and "lightweight" might invite the reader to think about a lack of security. However, here the combination refers to suitable cryptography for limited devices for which trade-offs between performance, security, and cost are highly important. The constraints could be computing power, memory, bandwidth, or vulnerability to attacks. Lightweight cryptography is not an alternative to traditional cryptography, it requires a change of perspective. Indeed, with the growth of ubiquitous devices, protecting security and privacy has become more important than ever. The paper focuses on describing lightweight-cryptography solutions for RFID tags. It examines possible risks and the measures researchers have taken to improve their level of security and privacy.

Note. Published in *IEEE Security and Privacy 2012*. A reworked and updated variant of this paper is included at the beginning of Chapter 5.

Authenticated Encryption: Toward Next-Generation Algorithms [MR14]

with Reza Reyhanitabar

Abstract. Do researchers have a cryptographic tool able to provide both confidentiality (privacy) and integrity (authenticity) of a message? The answer to that question is affirmative: authenticated encryption (AE), a symmetric-key mechanism that transforms a message into a ciphertext and authenticates it. This article discusses standard AE algorithms, classic security models' shortcomings for AE algorithms, and related attacks. Motivated by these attacks, the cryptographic community started a Competition for Authenticated Encryption: Security, Applicability, and Robustness to promote the development of next-generation AE algorithms.

Note. Published in *IEEE Security and Privacy 2014*. A reworked version of this paper is included in Chapter 3.

CHAPTER 3

MATHEMATICAL AND CRYPTOGRAPHIC PRELIMINARIES

Theory is important, at least in theory.
Keith Martin.

Summary

The underlying theoretical foundations of this thesis are presented within this chapter.

We start by presenting the notation used throughout the entire thesis. In view of the contributions described in Section 4.3, basic notions about hash functions and message authentication codes are presented in Section 3.1.

Section 3.2 provides basic information regarding Authenticated Encryption (AE): both the generic composition paradigm and the one-pass solution. Standard AE modes of operation are addressed, emphasising GCM in view of a comparison with the result in Section 4.3. Security notions are further presented.

Digital signature schemes are overviewed in Section 3.3. We discuss general concepts, shortly present a number of widely known signature schemes and recall representative security notions.

Elliptic curves are introduced in Section 3.4 and their importance in cryptography is discussed. Terminology is given and basic algorithms for the arithmetic over ECs are presented.

Notation. The set of all n-bit binary strings (for some $n \in \mathbb{N}^*$) is denoted as $\{0,1\}^n$, the set of all binary strings whose lengths are variable but upper-bounded by n is denoted as $\{0,1\}^{\leq n}$ and the set of all binary strings of finite length is denoted by $\{0,1\}^*$.

$x||y$ and xy denote the string obtained by concatenating y to x. For an m-bit binary string $x = x_{m-1} \cdots x_0$ we denote:

$$\text{MSB}\,(x) = x_{m-1} \quad \text{and} \quad \text{LSB}\,(x) = x_0\,.$$

For two binary strings $x = x_{m-1} \cdots x_0$ and $y = y_{n-1} \cdots y_0$, the notation $x \oplus y$ denotes bitwise XOR of $x_{m-1} \cdots x_{m-1-\ell}$ and $y_{n-1} \cdots y_{n-1-\ell}$ where $\ell = \min\{m-1, n-1\}$.

3.1 Hash Functions and Message Authentication Codes

Authentication is one of the most important information security goals. Confidentiality and authentication used to be considered as intrinsically connected until the 1970s.

The usefulness of hash functions for digital signature schemes was initially pointed out by Diffie and Hellman. Rabin [Rab79], Yuval [Yuv79] and Merkle [Mer79] were the first to provide definitions, analysis and constructions of cryptographic hash functions during the 1970s. Rabin proposed a 64-bit hash function based on the DES. Yuval focused on the analysis part, showing how to apply the birthday paradox to find collisions in hash functions. Merkle introduced the fundamental definitions of collision resistance, pre-image resistance, and second pre-image resistance. These properties are recalled later on.

Among commonly used and known hash functions nowadays, we mention the MD family [Riv] (*e.g.* MD2 and MD5), the Secure Hash Algorithms (SHA) which include the SHA-1, SHA-2 and, more recently, SHA-3, the RIPEMD family, HAVAL and Whirpool [1].

Hash functions are constructed from so-called "compression functions" [2]. A compression function takes a fixed length input and returns a shorter, usually fixed-length output [3]. A message of arbitrary length is divided into blocks and padded so that the size of the message is a multiple of the block size [4]. The blocks are processed sequentially, taking as input the result of the hash up to that point plus the current message block. The final output is the digest of the message.

We make use of SHA-2 in Section 4.3 and therefore recall the corresponding compression functions in Appendix A.

Birthday Attacks. The birthday paradox (problem) [vM39] familiarly refers to the probability that in a set of n randomly chosen people, two of them have the same birthday.

The **generalized birthday problem**: Given a year with x days, the generalized birthday problem asks for the minimal number $y(x)$ such that, in a set of y randomly chosen persons, the probability of a birthday coincidence is at least $\frac{1}{2}$. In other words, $y(x)$ is the minimal integer y such that

$$1 - \prod_{i=1}^{y-1} \left(1 - \frac{i}{x}\right) \geq \frac{1}{2}\,.$$

The most common numerical example is $x = 365$ (a year) for which $y = 23$ (persons) yields a probability of at least $\frac{1}{2}$.

Birthday attacks leverage this "paradox" and provide a time-memory collision-search trade-off [5]. A birthday attack on a 160-bit digest requires 2^{80} computations and 2^{80} storage. However, collisions can be computed using much less memory [QD90, vOW99].

1. [FIP12, BDPVA09, Bos11, ZPS92]
2. The compression function is applied repeatedly until the entire message is processed.
3. usually called *digest*
4. The padding field typically contains the total message length in bits.
5. Time-memory trade-off refers to saving memory at the cost of cryptanalysis time.

Hash Functions

Let $\texttt{Alph}$ denote an alphabet. H is a mapping $H : \texttt{Aplh}^* \to \texttt{Alph}^\ell$, where $\ell \in \mathbb{N}$ is usually a fixed integer [6]. Commonly, $\texttt{Alph} = \{0,1\}$ and $128 \leq \ell \leq 512$.

A natural requirement is that H should be easy to evaluate for any sequence in $\texttt{Alph}^*$. To make H cryptographically secure, cryptographers often require one or more of the following properties to hold.

1. **Pre-image resistance:** H is a *one-way* or *pre-image resistant* function if for almost all outputs b it is computationally infeasible to find an input $a \in \texttt{Alph}^*$ such that $b = H(a)$.
2. **Second pre-image resistance:** H is *weak collision resistant* or *second pre-image resistance* if given value $a \in \texttt{Alph}^*$ it is computationally infeasible to find a second value $a' \in \texttt{Alph}^*$, $a \neq a'$, such that $H(a) = H(a')$.
3. **Collision resistance:** H is *strong collision resistant* or, simply, *collision resistant* if it is computationally infeasible to find a pair of values $(a, a') \in (\texttt{Alph}^*)^2, a \neq a'$ such that $H(a) = H(a')$.

Hash functions are most commonly used in combination with digital signatures to reduce the number of calls to the (costly) signature algorithm. Usually a long message is hashed by means of a publicly available hash function and only the corresponding hash value is signed. The receiver re-hashes the message and verifies that the received signature is correct for this particular hash value. However, hash functions are also used for authenticating a short string (commitments and data authentication). Digital signatures are not needed in such an approach.

Even if a hash function meets the above properties, it is still possible to intercept a transmission $(a, H(a))$ and replace it by an $(a', H(a'))$ for some $a' \neq a$. Thus, in some specific cases, introducing a secret key (shared by the sender and the receiver) is desirable. Such constructions are called *Message Authentication Codes* (MACs, see Section 3.1 for a description of this concept).

Using block ciphers to construct keyless hash functions is possible.

Message Authentication Codes

Message authentication code algorithms (MAC algorithms) prevent an adversary from modifying messages sent by Alice to Bob, without Bob detecting the modification.

MAC algorithms are also referred to as keyed hash functions. A good way of constructing MAC algorithms is by means of a pseudo-random function (PRF). A common class of MAC algorithms is obtained via the cipher-block chaining mode (CBC) applied for a given block-cipher [fip85]. [fip85] was withdrawn in September 2008 because it was based on the obsolete DES.

Defining a MAC. A MAC is a 3-tuple of PPT algorithms: (KEYGEN, MAC, VERIFY). KEYGEN is a probabilistic algorithm. VERIFY consists in re-running MAC on the same input and comparing the outputs of MAC and VERIFY.

The descriptions of these three algorithms are given in Table 3.1.

KEYGEN	Let κ be the security parameter and let 1^κ be the input of the key generation algorithm KEYGEN. KEYGEN outputs a key k.
MAC	Given a message m and a key k, MAC [7] outputs a tag t.
VERIFY	Given k, m, t, VERIFY re-runs MAC and tests if t is a valid tag of m with respect to k.

Table 3.1: Algorithms of a MAC.

Definition 3.1 (The Message Authentication Game MAC-FORGE$_{\mathcal{A},\mathbf{MAC}}(\kappa)$)

6. Recently, NIST standardized SHAKE-256 and SHA-512, which are extendible output functions based on the Keccak hash function design.
7. The MAC algorithm may be randomized.

1. *A random key* $k \xleftarrow{\$} \{0,1\}^\kappa$ *is chosen.*
2. *The adversary* $\mathcal{A}$ *is given oracle access to* $\text{MAC}_k(\cdot)$ *and outputs a pair* (m,t)*. Formally,* $(m,t) \leftarrow \mathcal{A}^{\text{MAC}_k(\cdot)}(1^\kappa)$*. Let* A *denote the queries of* $\mathcal{A}$ *to the oracle* $\text{MAC}_k(\cdot)$*.*
3. *The output after performing the above steps is defined to be* $1 \Leftrightarrow m \notin A$ *and* $\text{VERIFY}(m,t) = \texttt{True}$*.*

Definition 3.2 (MAC-FORGE**)** *A MAC is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* f *such that:* $\Pr[\text{MAC-FORGE}_{\mathcal{A},MAC}(\kappa) = 1] \leq f(\kappa)$.

3.2 Authenticated Encryption (AE)

With the progressive advent and formalization of hash functions and digital signatures, the fact that confidentiality and authentication are truly separate and independent objectives became progressively apparent.

3.2.1 From the Generic Composition Paradigm to Dedicated AE Algorithms

Authenticated Encryption (AE) is a symmetric-key mechanism, a tool able to provide both confidentiality (privacy) and integrity (authenticity).

Generic Composition. Originally, AE algorithms achieved confidentiality and integrity by combining two separate primitives: a conventional encryption algorithm to ensure confidentiality and a MAC to guarantee integrity. However, this approach is computationally suboptimal (because parties process the input message at least twice) and prone to implementation errors stemming from the miscombining of encryption and authentication mechanisms.

Dedicated Solutions. To address these concerns, researchers developed AE algorithms that function as a desirable primitive that APIs would offer to end developers. Providing direct access to a single AE functionality (rather than requiring developers to call two different lower-level functionalities) as a step toward improving security-critical code.

We refer the reader to [Pre93] for an overview of the solutions studied in the 1980s [8].

The next sections address standard AE algorithms, classic security models' shortcomings for AE algorithms, related attacks and a recent AE design competition in which we participated.

Authenticated Encryption as a Cryptographic Goal

AE is adopted by many widely implemented standards [9] such as Secure Shell (SSH), Secure Sockets Layer/Transport Layer Security (SSL/TLS), IPsec, Wi-Fi and Smart Grid standards to quote only a few. This wide range of applications made AE ubiquitous in modern security protocols. Developing and analysing efficient and provably secure AE algorithms are hence important research goals. Compared to traditional encryption algorithms and MACs, AE algorithms are relatively new. Many still think that privacy and integrity can be straightforwardly achieved, by simply combining a traditional encryption algorithm and a MAC: encrypt-and-MAC, MAC-then-encrypt and encrypt-then-MAC are three such attempts. The AE algorithms adopted by many standards (*e.g.* SSH, SSL/TLS, and IPsec) fall into this category.

Unfortunately, miscombinations of these algorithms resulted in several successful attacks [BKN04, Vau02]. For example, a message recovery attack against OpenSSH-encrypted plaintexts appeared in 2009 [APW09]. The BEAST (Browser Exploit Against SSL/TLS) attack, which uses a vulnerability in TLS 1.0, appeared in 2011 [DR11]. Most of these attacks targeted generic composition algorithms (encrypt-and-MAC and MAC-then-encrypt), but a recent attack against EAX-Prime shows that dedicated AE algorithms are also very subtle mathematical objects where precise assumptions are key [MLMI12].

Using secure authenticated encryption schemes would have thwarted some of the most visible recent attacks. Developing models that include implementation errors and other deviations from mathematical abstractions is an important aspect of cryptographic primitive evaluation. Hence, a better understanding of AE schemes must be set as a goal.

8. *e.g.* Jueneman's schemes consequently broken by Coppersmith.
9. [rfc06, rfc08, FKL+05, IEE99, SKD+15]

Standard AE Modes of Operation

Several standardization bodies adopted a number of AE modes of operation.

The US National Institute of Standards and Technology (NIST) approved five (block cipher) modes for confidentiality and authentication [WHF03]. One example is CCM (Counter with Cipher Block Chaining Message Authentication Code). The specification is intended to be compatible with using CCM within a draft amendment to IEEE 802.11. Another example is GCM (Galois Counter Mode). During the selection process, another NIST candidate was EAX [BRW03]. These three modes are AEAD (AE with Associated Data) algorithms. IEEE standards for AE also include CCM and GCM.

ISO/IEC 19772 standardized six AE modes. These include the three modes that we just mentioned and OCB 2.0 (Offset Codebook Mode) which is a scheme that integrates an MAC into a block cipher [KR14].

The Internet Engineering Task Force RFC 5297 [Har08] describes Synthetic Initialization Vector (SIV) mode. SIV can work for both conventional (nonce-based, misuse-resistant) and deterministic AE.

The next paragraph focuses on presenting the basics of the Galois Counter Mode (GCM). We chose this authenticated mode of operation in view of the OMD authenticated encryption scheme, one of our thesis results detailed in Section 4.3. Considering various features of AES-GCM, we give a comparison between AES-GCM and our scheme, OMD.

Galois Counter Mode (GCM). The Galois Counter Mode (GCM) is a block cipher mode of operation based on universal hashing over a binary Galois field in order to provide authenticated encryption.

GCM may be considered as a stand-alone MAC and can be used with initialization vectors (IVs) of arbitrary length. GCM consists of two operations, namely authenticated encryption and authenticated decryption.

Algorithm 3.2 presents the steps of the GCM encryption. Algorithm 3.3 describes the corresponding decryption process. Function GHASH is defined by the formula $\text{GHASH}(H, A, C) = X_{m+n+1}$, where the variables X_i for are defined as in Algorithm 3.1.

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, ..., m-1 \\ (X_{m-1} \oplus (A_m^* || 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_i) \cdot H & \text{for } i = m+1, ..., m+n-1 \\ (X_{m+n-1} \oplus (C_m^* || 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\text{len}(A) || \text{len}(C))) \cdot H & \text{for } i = m+n+1 \end{cases} \tag{3.1}$$

$$\text{Algorithm ENCRYPTION} = \begin{cases} H & \leftarrow & \text{ENC}(K, 0^{128}) \\ Y_0 & \leftarrow & \begin{cases} IV || 0^{31}1, \text{ if len}(IV) = 96 \\ \text{GHASH}(H, \{\}, IV), \text{ otherwise} \end{cases} \\ Y_i & \leftarrow & \text{incr}(y_{i-1}) \text{ for } i = 1, ..., n \\ C_i & \leftarrow & P_i \oplus \text{ENC}(K, Y_i) \text{ for } i = 1, ..., n-1 \\ C_n^* & \leftarrow & P_n^* \oplus \text{MSB}_u(\text{ENC}(K, Y_n)) \\ T & \leftarrow & \text{MSB}_t(\text{GHASH}(H, A, C) \oplus \text{ENC}(K, Y_0)) \end{cases} \tag{3.2}$$

$$\text{Algorithm DECRYPTION} = \begin{cases} H & \leftarrow & \text{ENC}(K, 0^{128}) \\ Y_0 & \leftarrow & \begin{cases} IV||0^{31}1, \text{ if len}(IV) = 96 \\ \text{GHASH}(H, \{\}, IV), \text{ otherwise} \end{cases} \\ Y_i & \leftarrow & \text{incr}(y_{i-1}) \text{ for } i = 1, ..., n \\ T & \leftarrow & \text{MSB}_t(\text{GHASH}(H, A, C) \oplus \text{ENC}(K, Y_0)) \\ P_i & \leftarrow & C_i \oplus \text{ENC}(K, Y_i) \text{ for } i = 1, ..., n-1 \\ P_n^* & \leftarrow & C_n^* \oplus \text{MSB}_u(\text{ENC}(K, Y_n)) \end{cases} \tag{3.3}$$

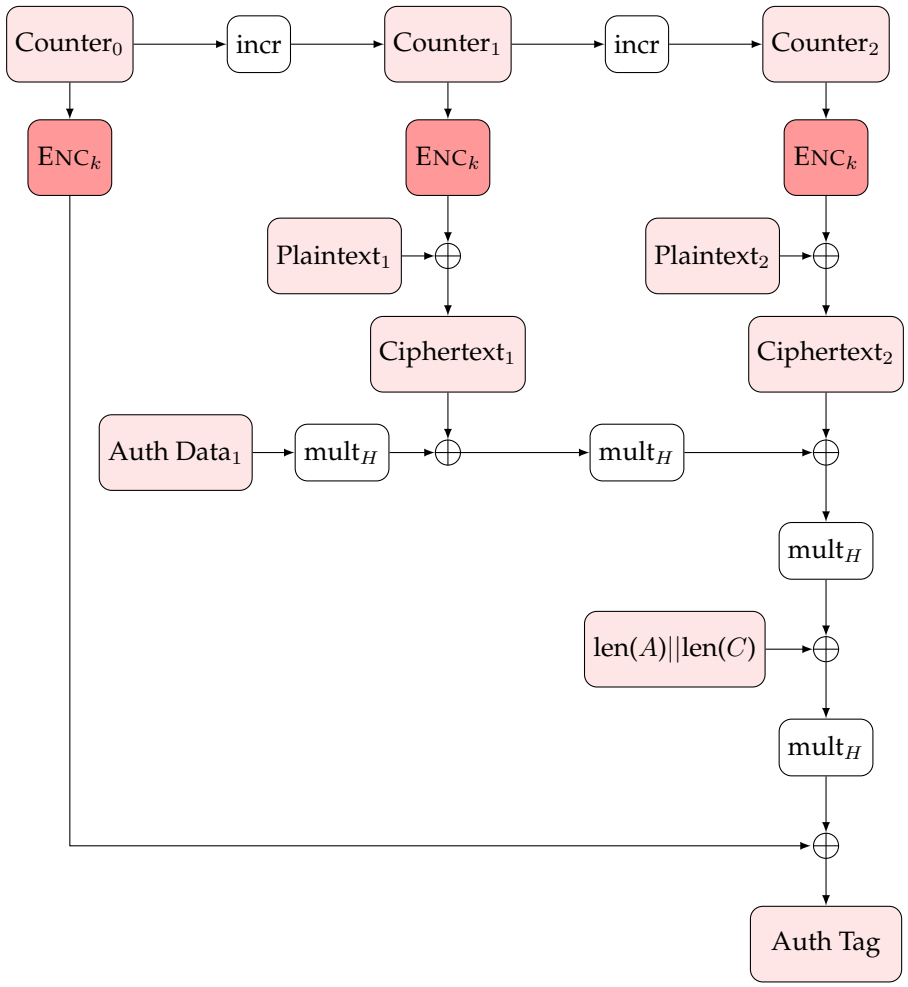


Figure 3.1: The GCM authenticated mode of operation - ENCRYPT.

Security Models for Authenticated Encryption

Although AE security has received much attention, several issues remain. For example, additional security and robustness goals are often left as an option. Also, no consensus exists on a set of comprehensive security properties that would capture a variety of real-world attack scenarios.

The current AE security definitions presume that the entire ciphertext is offered for decryption and a whole plaintext (or error message) is returned [BN08, RS06]. However, in some applications, the

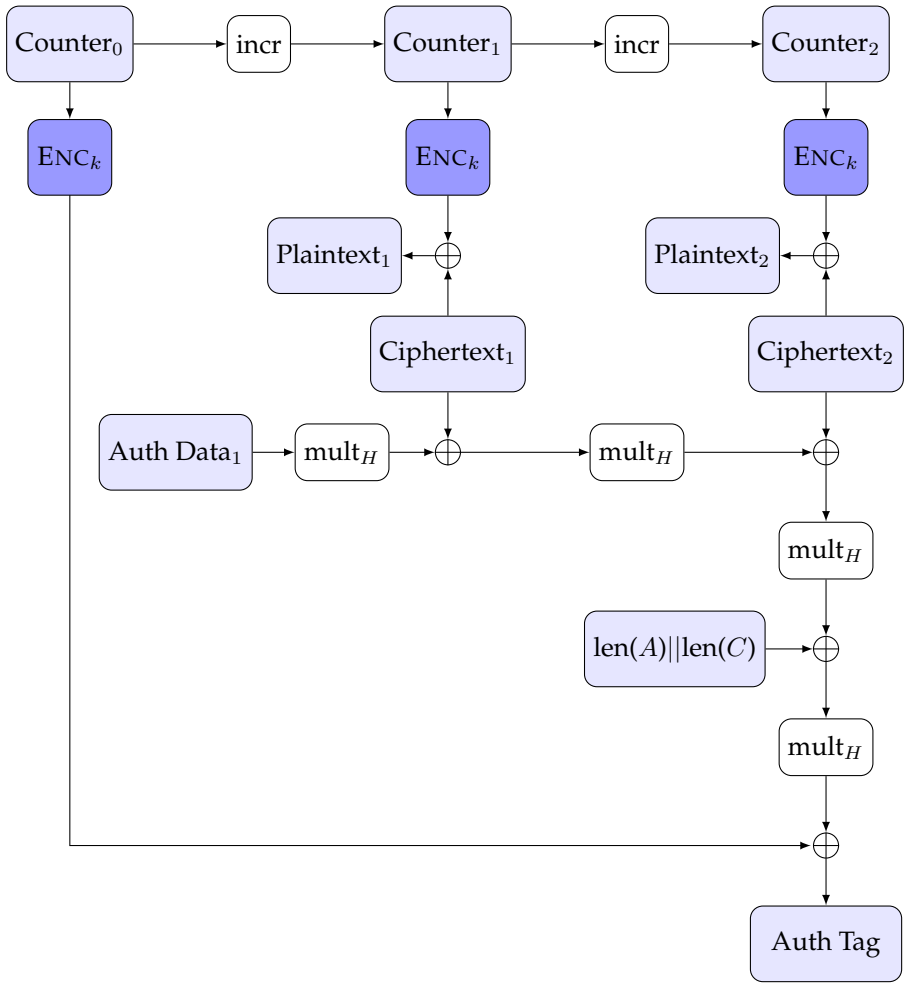


Figure 3.2: The GCM authenticated mode of operation - DECRYPT.

encrypted data is delivered to the receiver block-by-block. This leads to attacks that bypass the security models' plaintext/ciphertext atomicity assumption [JMV02, DR11, BDPS12].

Current AE security models don't capture many real-world threats such as information leakage via software and hardware side channels [10] [11]. Vaudenay identified this weakness [Vau02] of AE security models. He described side channel attacks that used error information leaked in the padding verification to attack the MAC-then-encrypt composition used in SSL and many other protocols.

Conventional algorithms feature indistinguishability and nonmalleability as defined in [GM84]. In addition, Bellare and Namprempre [BN08] formalized different notions of integrity together with their relationships in the AE context. Up to now, we discussed AE algorithms with a stateful or probabilistic encryption process but with a stateless deterministic decryption algorithm. Bellare *et al.* [BKN04] extended previous models to include stateful decryption, which can thwart more real-world threats, such as replay attacks and out-of-order-delivery attacks.

Previous AE ciphers modelled encryption and decryption as two-input algorithms (one argument for the plaintext or ciphertext and the other for the secret key). This approach assumes that the algorithms will generate and handle any randomness or state information. Rogaway [Rog02] extended the AE syntax to include a third input for associated data (AD). This aims to model the practical scenarios in which part of the message (for example, some header information) needs integrity protection but not privacy protection. AD is assumed to be provided in clear to both the encryption and decryption algorithms.

Nonce-based AE makes the critical assumption that no nonce will repeat during a secret key's lifetime [Rog04a]. However, nonce reuse (which is, in fact, a misuse) may occur in practice. To address the

10. Peter Wright described side channels in [Wri87].
11. Kocher was the first to publish timing attacks [Koc96].

threats that stem from such a reuse, Rogaway and Shrimpton [RS06] proposed deterministic AE (DAE) and provided a secure DAE scheme based on the generation of the nonce from the message. This technique is currently called SIV (Synthetic Initialization Vector). Fleischmann *et al.* noted that the known DAE schemes weren't computable online, contrary to many applications' requirements [FFL12]. Hence [FFL12] introduced on-line AEAD and a secure construction achieving online computability.

A careful review of the state of the art reveals an expanding list of discrepancies between conventional AE security models and the features desired by real-world applications scenario. Two problems of particular interest involve formalizing concise security models for AE algorithms that resist fragmentation attacks [Rog04a] and side channel attacks [FPS12].

We will now formalize the security notions related to AE schemes, following the security notions given in [Rog02].

Nonce Respecting Adversaries. Let $\mathcal{A}$ be an adversary. Let N denote nonces, M denote messages and $\mathbb{C} = C||\text{Tag}$, where C are ciphertexts. We say that an attacker $\mathcal{A}$ is nonce-respecting if $\mathcal{A}$ never repeats a nonce in its *encryption* queries. That is, if $\mathcal{A}$ queries the encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ on $(N_1, A_1, M_1), \cdots, (N_q, A_q, M_q)$ then $N_1, \cdots, N_q$ must be pairwise distinct.

In the following, we define the conventional AEAD security properties: namely, privacy (*confidentiality for the plaintext*) and authenticity (*integrity for the nonce, for the associated data, and for the plaintext*).

Privacy of AEAD Schemes. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AEAD scheme. Let $\mathcal{A}$ be a nonce-respecting adversary. $\mathcal{A}$ is provided with an oracle which can be either a real encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ returning $\mathbb{C} = \mathcal{E}_K(N, A, M)$ on input (N, A, M), or a fake encryption oracle $\$(\cdot,\cdot,\cdot)$ which on any input (N, A, M) returns $|\mathbb{C}|$ fresh random bits. The advantage of $\mathcal{A}$ in mounting a chosen plaintext attack (CPA) against the privacy property of Π is defined as follows:

$$\text{Adv}^{\text{priv}}_{\Pi}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} = 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} = 1].$$

This privacy notion, also called indistinguishability of ciphertext from random bits under CPA (IND$-CPA), was introduced by [RBBK01] and is a stronger variant of the classical IND-CPA notion [BDJR97, BN00] for conventional symmetric-key encryption schemes.

Resource Parameters for the CPA Adversary. Let the CPA-adversary $\mathcal{A}$ make queries $(N_1, A_1, M_1), \cdots, (N_{q_e}, A_{q_e}, M_{q_e})$. We define the resource parameters of $\mathcal{A}$ as $(t, q_e, \sigma_A, \sigma_M, L_{\max})$ where:

- t is the time complexity
- q_e is the total number of encryption queries
- $\sigma_A = \sum_{i=1}^{q} |A_i|$ is the total length of associated data in bits
- $\sigma_M = \sum_{i=1}^{q} |M_i|$ is the total length of messages in bits
- $L_{\max}$ is the maximum length of each query in bits.

The absence of a resource parameter means that the parameter is irrelevant in the context and is hence omitted.

Authenticity of AEAD Schemes. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a nonce-based AEAD scheme. Let $\mathcal{A}$ be a nonce-respecting adversary. We stress that nonce-respecting only refers to encryption queries; *i.e.*, $\mathcal{A}$ may repeat nonces during its decryption queries. $\mathcal{A}$ may also ask an encryption query with a nonce that was already used in a decryption query. Let $\mathcal{A}$ be provided with the encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ and the decryption oracle $\mathcal{D}_K(\cdot,\cdot,\cdot)$; *i.e.*, we consider adversaries that can mount chosen ciphertext attacks

(CCA). We say that $\mathcal{A}$ forges if it makes a decryption query $(N, A, \mathbb{C})$ such that $\mathcal{D}_K(N, A, \mathbb{C}) \neq \perp$ and no previous encryption query $\mathcal{E}_K(N, A, M)$ returned $\mathbb{C}$.

$$\mathrm{Adv}^{\mathrm{auth}}_{\Pi}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot),\, \mathcal{D}_K(\cdot,\cdot,\cdot)} \text{ forges}] .$$

This authenticity notion, also called ciphertext integrity (INT-CTXT) under CCA attacks, was defined in [BN00] [12].

Resource Parameters for the CCA Adversary. Let the CCA-adversary $\mathcal{A}$ make encryption queries $(N_1, A_1, M_1), \cdots, (N_{q_e}, A_{q_e}, M_{q_e})$ and decryption queries $(N'_1, A'_1, \mathbb{C}'_1), \cdots, (N'_{q_v}, A'_{q_v}, \mathbb{C}'_{q_v})$. We define the resource parameters of $\mathcal{A}$ as $(t, q_e, q_v, \sigma_A, \sigma_M, \sigma_{A'}, \sigma_{\mathbb{C}'}, L_{\max})$, where:

- t is the time complexity
- q_e and q_v are respectively the total number of encryption queries and the number of decryption queries
- $L_{\max}$ is the maximum size of each query in bits
- $\sigma_A = \sum_{i=1}^{q_e} |A_i|$
- $\sigma_M = \sum_{i=1}^{q_e} |M_i|$
- $\sigma_{A'} = \sum_{i=1}^{q_v} |A'_i|$
- $\sigma_{\mathbb{C}'} = \sum_{i=1}^{q_v} (|\mathbb{C}'_i| - \tau)$.

The omission of a resource parameter means that the parameter is irrelevant in the context.

Note. The use of the aforementioned privacy (IND-CPA) and authenticity (INT-CTXT) goals to define AE security schemes is credited to [BN00] who showed that if an AE scheme is IND-CPA secure and INT-CTXT secure then this AE scheme will also be IND-CCA which, in turn, is equivalent to NM-CCA.

Note. The nonce-respecting assumption on the adversary is justified as follows. The nonce would typically be a counter (message number) maintained by the sender who encrypts the messages. In practice, an implementation must make sure that no nonce gets repeated within a session (*i.e.*, the lifetime of the current encryption key). As the nonce N is needed both to encrypt and to decrypt, N would be typically transmitted *in clear*. Note that nonce-respecting is only assumed with respect to encryption queries, reflecting the fact that the sender who encrypts a message is the party responsible for providing fresh nonces and the receiver may be stateless.

On Some Attacks. Verifying the provided security proofs and their underlying assumptions is key and flawed proofs aren't uncommon [BKN04]. For instance, Iwata *et al.* showed that GCM's original proofs are flawed [IOM12]. Iwata *et al.* presented a fixed proof, but the new security bounds turned out to be worse than the previously claimed ones. This shows that even if the target algorithm's original proofs are verified and certified, a detailed re-examination may degrade (if a mistake is spotted) or improve the bounds.

Besides the conventional analyse of the schemes' security, it is very important to challenge the *models* in which proofs are given. Are of particular interest the flaws that might arise when AE schemes operate in environments that don't meet the mathematical abstractions used in the proofs. For example, Bellare *et al.* proved that SSH BPP [13] variants variants were secure [DR11], using a traditional security model treating plaintexts and ciphertexts as atomic. However, Albrecht *et al.* broke SSH BPP in 2009, exploiting the fact that encrypted data can be delivered to the receiver in a fragmented manner [APW09]. The analysis of AE security in such advanced threat models must consider a variety of practical misuses. Potential ways

12. not yet defined in the nonce-based context
13. binary packet protocol

to contradict and violate a scheme must thus be permanently investigated and compared to real-world scenarios.

Andreeva *et al.* [And14] have addressed the *releasing unverified plaintext* [14] (RUP) setting in authenticated encryption.

14. outputting decrypted plaintext before verification

3.3 Digital Signatures

Digital signatures can be thought of as the public-key equivalent of MACs. By opposition to MACs, digital signatures have the advantage of being publicly verifiable and non-repudiable. Public *verifiability* implies the *transferability* of signatures and, thus, signatures prove useful in many applications and in public-key infrastructures. *Non-repudiation* and verifiability make digital signatures particularly suitable for contract signing purposes. Contract signing will be analysed in further depth in Section 4.1.

Defining a Digital Signature Scheme. A digital signature scheme Σ is a 3-tuple of algorithms: (KEYGEN, SIGN, VERIFY). KEYGEN is probabilistic algorithm. SIGN is usually probabilistic, but may, in some cases, be deterministic. VERIFY is usually deterministic.

The descriptions of these three algorithms are given in Table 3.2.

KEYGEN	Let k be the security parameter and let 1^k be the input of the key generation algorithm KEYGEN. KEYGEN outputs a pair (pk, sk) of public and secret keys.
SIGN	Given a message m and (pk, sk), SIGN outputs a signature σ.
VERIFY	Given σ, m, pk, VERIFY tests if σ is a valid signature of m with respect to pk.

Table 3.2: Algorithms of a digital signature scheme.

The correctness of a digital signature scheme is defined as follows.

Definition 3.3 (Correctness of a Signature Scheme) *Let $\mathcal{M}$ be a message space A signature scheme is said correct if, for any message $m \in \mathcal{M}$ the following experiment:*

$$(sk, pk) \leftarrow \text{KEYGEN}(\lambda), \quad \sigma \leftarrow \text{SIGN}(sk, m), \quad b \leftarrow \text{VERIFY}(pk, m, \sigma)$$

is such that $b =$ `True` except with probability negligible in λ.

3.3.1 General Concepts

Consider a digital document m, *e.g.* a text file. The document's author wishes to add to m some extra information σ such that σ could serve as a proof of the authors' approval of m or m's origin. Hence, it is easy to see an analogy between standard (physical) signatures and their computer-age relatives, digital signatures. Note that physical signatures must also be verifiable.

A signature, be it physical or digital, must be unique, non-imitable, easy to verify, undeniable and easy to generate.

Formally, Alice's signature on a message m is a string σ depending on m, on user-specific public and secret data and (possibly) on a randomiser. Anyone can check the validity of σ using only public data.

The public information pk is called the *public key*. The secret key is usually denoted by sk. Signatures must be impossible to produce (forge) by entities who do not possess sk.

Definition 3.4 (Trapdoor Permutation Family) *A trapdoor permutation family is a tuple of PPT algorithms in the security parameter (*GEN, EVAL, INVERT*) such that:*

1. GEN(1^k) *outputs a pair (f, f^{-1}), where f is a permutation over $\{0,1\}^k$.*
2. *Assume f was output by* GEN *and $x \in \{0,1\}^k$.* EVAL$(1^k, f, x)$ *is a deterministic algorithm which outputs $y \in \{0,1\}^k$.*
3. *Assume f^{-1} was output by* GEN *and $y \in \{0,1\}^k$.* INVERT$(1^k, f^{-1}, y)$ *is a deterministic algorithm which outputs $x \in \{0,1\}^k$.*
4. **Correctness***: For all k, all (f, f^{-1}) output by* GEN *and all $x \in \{0,1\}^k$, we have $f^{-1}(\text{EVAL}(1^k, f, x)) = x$.*
5. **One-wayness***: For all PPT algorithms A, the following is negligible:*

$$Pr[(f, f^{-1}) \leftarrow \text{GEN}(1^k); y \leftarrow \{0,1\}^k; x \leftarrow A(1^k, f, y) : f(x) = y] \,.$$

A method allowing to construct signatures from encryption trapdoor permutations was presented in Diffie and Hellman's paper [DH76]. The first signature scheme based on such a trapdoor permutation was the celebrated RSA algorithm, proposed in 1978 by Rivest, Shamir and Adleman in [RSA78].

A new method of obtaining signature schemes (derived from fair zero-knowledge identification protocols [15]) was introduced by Goldwasser, Micali and Rackoff in 1986 [GMR85].

Fiat and Shamir [FS87] constructed a zero-knowledge identification protocol whose underlying security assumption was the hardness of extracting modular square roots. [FS87] also presents a corresponding signature scheme and discusses its security at length.

The first RSA-based digital signature international standard appeared in 1991 [Sec91].

As time passed different types of signatures with additional properties appeared. Five such examples are blind signatures [Cha82], designated verifier signatures [JSI96a], group signatures [RST01], ring signatures [RST01] and autotomic signatures [NP12] to quote a few. Fully distributed signatures are addressed later on, in Section 4.1.

Public Key Infrastructure (PKI). In his very original Bachelor thesis [Koh78], Kohnfelder established very important concepts of PKI fot the first time.

A *public key infrastructure* (PKI) is a system allowing the distribution and the identification of public-keys. PKIs enable users to securely exchange data over networks and verify the identities of remote parties. Digital certificates are at the core of PKIs. Certificates [16] confirm the identity of a party, and connect a user identity with his public key contained in the certificate.

A typical PKI includes:
- A trusted party called *certificate authority* (CA), acting as a root of trust
- A *registration authority* (RA). In some systems, an RA is a "subordinate CA", certified by the CA to issue lower-level certificates; nevertheless, in other systems an RA records users and public keys but not sign certificates.
- A certificate database, storing certificate requests, issuing and revoking certificates
- A certificate store, residing on client computers and storing certificates, public keys and private keys

The CA or an RA issues digital certificates to entities and individuals after verifying their identities. The CA signs certificates using its private key. The CA's public key is available to all in a self-signed CA certificate. RAs use the CA to create a "chain of trust". As we write these lines, many root certificates are embedded in Web browsers [17]. Certificates contain information such as the signing algorithms, entity identities, expiry dates and entity security privileges.

3.3.2 Signature Schemes

A number of well known digital signature algorithms will further be recalled schematically: RSA in Figure 3.3, ElGamal in Figure 3.4, Schnorr in Figure 3.5 and Girault-Poupard-Stern in Figure 3.6. We present the ECDSA in Section 5.6.

The reason why we chose to present the ElGamal signatures is not only historical. Both ElGamal and Girault-Poupard-Stern can serve to implement the construction described in Section 4.1.

Also, Schnorr signatures are of particular interest for the results of Section 4.1, where Schnorr co-signature for two parties is described and proved.

An important result about digital signature schemes is that any public-key encryption scheme for which $\mathcal{C} = \mathcal{M}$, can be used as a digital signature scheme.

15. For the sake of precise paternity attribution the GMR construction is implicitly used by El Gamal in [EG84].
16. which are nothing but signatures on public-keys and associated data
17. Thus, having built-in trust of those CAs.

RSA (Rivest-Shamir-Adleman)

RSA is probably the most popular signature scheme to date. RSA signature and encryption share the same key generation algorithm. The security of both relies on the FACT assumption (*cf.* Section 2.1.2). A variety of RSA-based signature schemes appeared over time. Despite its elegant mathematical structure, instantiating RSA is subtle as shown in [Mis98, CNS99, CND+06].

Parameters and Key Generation	
	Choose large primes p, q and compute $n = p \cdot q$
	Choose e such that $\gcd(e, \varphi(n)) = 1$,
	where $\varphi(\cdot)$ is Euler's totient function [18]
Private	d such that $e \cdot d = 1 \bmod \varphi(n)$
Public	n, e
Signing Algorithm (message m)	
	$\sigma \leftarrow m^d \bmod n$
	Output σ as the signature of m
Verification Algorithm [19]	

	if $m = \sigma^e \bmod n$ **then**
	  return `True`
	else
	  return `False`

Figure 3.3: RSA signature algorithm.

ElGamal

ElGamal's digital signature and encryption algorithms were introduced in [EG84]. [EG84] didn't include the hashing step added in Figure 3.4. Pointcheval-Stern's version of ElGamal signatures is provably secure against adaptive chosen-message attacks [PS96, PS00].

Nonetheless, Bleichenbacher [Ble96] has shown how malicious parameters can be generated. Bleichenbacher's attack is also applicable to Pointcheval-Stern's version of ElGamal.

Schnorr

Schnorr signatures [Sch90] are an ElGamal offspring [EG84] signatures. Schnorr signatures are provably secure in the Random Oracle Model under the DLP assumption [PS96].

Schnorr signatures will be discussed in further detail in Section 4.1.

Girault-Poupard-Stern (GPS)

Originally, the Girault-Poupard-Stern (GPS) scheme [GPS06] was developed as an identification scheme whose underlying hardness assumptions were DLP and FACT (see Section 2.1.2 for corresponding definitions). Applying the Fiat-Shamir transformation [FS87], GPS can be turned into a digital signature scheme in a straightforward manner.

GPS was the only identification scheme submitted to the NESSIE competition [NES]. NESSIE ended with 17 selected algorithms (of the initial 42), amongst which was GPS. The main advantage of GPS over other DLP based schemes is that the prover has only one exponentiation and one addition to perform. The exponentiation can be precomputed before receiving the challenge. If this is done the prover does not need to perform any modular reductions after receiving the challenge from the verifier.

19. Let a be a positive integer. Euler's totient function denoted above $\varphi(a)$ represents the number of positive integers b such that $1 \leq b \leq a$ and $\gcd(a, b) = 1$.
19. For simplicity, we do not consider redundancy check in m for this signature scheme.

Parameters and Key Generation	
	Large prime p such that $p-1$ contains a large prime factor
	$g \in \mathbb{Z}_p^*$.
Private	$x \in_R \mathbb{Z}_p^*$
Public	$p, g, y \leftarrow g^x \bmod p$
Signing Algorithm (message m)	
	Pick $k \in_R \mathbb{Z}_p^*$ such that $\gcd(k, p-1) = 1$
	$r \leftarrow g^k \bmod p$
	$e \leftarrow H(m)$
	$s \leftarrow e - xrk^{-1} \bmod p-1$
	If $s = 0$ repeat signature generation.
	Output $\{r, s\}$ as the signature of m
Verification Algorithm	
	$e \leftarrow H(m)$
	If $0 < r < p$ or $0 < s < p-1$ then return `False`
	if $g^e = y^r \cdot r^s \bmod p$ **then**
	return `True`
	else
	return `False`

Figure 3.4: ElGamal signature algorithm.

Parameters and Key Generation	
	Large primes p, q such that $q \geq 2^\kappa$, where
	κ is the security parameter and
	$p - 1 \bmod q = 0$
	$g \in \mathbb{G}$ (a cyclic group of prime order q)
Private	$x \in_R \mathbb{Z}_q^*$
Public	$y \leftarrow g^x$
Signing Algorithm (message m)	
	Pick $k \in_R \mathbb{Z}_q^*$
	$r \leftarrow g^k$
	$e \leftarrow H(m, r)$
	$s \leftarrow k - ex \bmod q$
	Output $\{r, s\}$ as the signature of m
Verification Algorithm	
	$e \leftarrow H(m, r)$
	if $g^s y^e = r$ **then**
	return `True`
	else
	return `False`

Figure 3.5: Schnorr's signature algorithm.

GPS is a version of Schnorr's signature algorithm designed to reduce on-line computation and thus allow on-the-fly signatures. The parameters of the GPS signature scheme have to be chosen taking into consideration the attack presented by van Oorschot and Wiener in [vOW96b].

Parameters and Key Generation	
	$A, B, S \in \mathbb{N}$ s.t. $\lvert A\rvert \geq \lvert S\rvert + \lvert B\rvert + 80, \lvert B\rvert = 32, \lvert S\rvert > 140$
	κ is the security parameter and
	p, q primes and $n = pq$
	$g \in_R \mathbb{Z}_n$ s.t. $\gcd(g, n) = 1$.
Private	$s \in [1, S]$
Public	$I = g^s \bmod n$
Signing Algorithm (message m)	
	Pick $r \in_R [0, A-1]$
	$x \leftarrow g^r \bmod n$
	$c \leftarrow H(m, x)$
	$y \leftarrow r + c \cdot s$
	Output $\{c, y\}$ as the signature of m
Verification Algorithm	
	if $0 < c < B-1$ or $0 < y < A + (B-1) \cdot (S-1) - 1$
	return `False`
	Compute $c' \leftarrow H(m, g^y / I^c \bmod n)$
	if $c' = c$ **then**
	return `True`
	else
	return `False`

Figure 3.6: Girault-Poupard-Stern's signature algorithm.

Security Notions for Digital Signatures

Bellare and Rogaway introduced the concept of "practice-oriented provable security" in a number of papers of which the first one was [BR94]. This concept naturally results from the convergence of theory (notably [GM84]) and practice.

As a direct consequence, the random oracle model (we refer the reader to the paragraph in Section 2.1.2) was introduced in [BR93]. The random oracle model allowed to prove the security of many cryptographic schemes by abstracting away the random-like properties of hash functions.

Using the random oracle model, Pointcheval and Stern [PS00] present security arguments for a large class of digital signatures. From their work emerged an approach that proposes computational reductions to well established problems for proving the security of digital signature schemes.

Security Notions. An efficient adversary $\mathcal{A}$ is modelled as a probabilistic polynomial time (PPT) algorithm.

We say that an adversary $\mathcal{A}$ "forges a signature on a new message" or "outputs a forgery" whenever $\mathcal{A}$ outputs a message/signature pair (m, σ) such that $\text{VERIFY}(pk, m, \sigma) = \texttt{True}$ and $\mathcal{A}$ was not previously given any signature on m. We say that $\mathcal{A}$ "outputs a strong forgery" whenever $\mathcal{A}$ outputs a message/signature pair (m, σ) such that $\text{VERIFY}(pk, m, \sigma) = \texttt{True}$ and $\mathcal{A}$ was not previously given the signature σ on the message m. Note that whenever $\mathcal{A}$ outputs a forgery then $\mathcal{A}$ also outputs a strong forgery.

Let Σ denote a signature scheme as defined in Section 3.3.

Definition 3.5 (EU-RMA Security) *A signature scheme Σ is* **existentially unforgeable under a random-message attack** *(EU-RMA) if for all polynomials p and all PPT adversaries $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following experiment is negligible, as a function of k:*

1. *A sequence of $p = p(k)$ messages $m_1, ..., m_p$ are chosen uniformly at random from the message space*
2. *$\text{KEYGEN}(1^k)$ is run to obtain a pair of keys (pk, sk)*
3. *Signature $\sigma_1 \leftarrow \text{SIGN}(sk, m_1), ..., \sigma_p \leftarrow \text{SIGN}(sk, m_p)$ are computed*

4. *$\mathcal{A}$ is given pk and $\{(m_i, \sigma_i)\}_{i=1}^{p}$ and outputs (m, σ)*
5. *$\mathcal{A}$ succeeds if* VERIFY$(pk, m, \sigma) =$ `True` *and* $m \notin (m_1, ..., m_p)$

Σ is **strongly unforgeable under a random-message attack** (SU-RMA) if Σ complies with Definition 3.5 where step 5 is replaced by

5. $\mathcal{A}$ succeeds if VERIFY$(pk, m, \sigma) =$ `True` and $(m, \sigma) \notin \{(m_1, \sigma_1), ..., (m_p, \sigma_p)\}$

Definition 3.6 (EU-KMA **Security)** *A signature scheme Σ is* **existentially unforgeable under a known-message attack** *(*EU-KMA*) if for all polynomials p and all PPT adversaries $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following experiment is negligible as a function of k:*

1. *$\mathcal{A}(1^k)$ outputs a sequence of $p = p(k)$ messages $m_1, ..., m_p$*
2. KEYGEN(1^k) *is run to obtain a pair of keys (pk, sk)*
3. *Signature $\sigma_1 \leftarrow$* SIGN$(sk, m_1), ..., \sigma_p \leftarrow$SIGN$(sk, m_p)$ *are computed*
4. *$\mathcal{A}$ is given pk and $\{\sigma_i\}_{i=1}^{p}$ and outputs (m, σ)*
5. *$\mathcal{A}$ succeeds if* VERIFY$(pk, m, \sigma) =$ `True` *and* $m \notin (m_1, ..., m_p)$

We assume that $\mathcal{A}$ is a stateful algorithm, and in particular is allowed to maintain state between steps 1 and 4.

Σ is **strongly unforgeable under a known-message attack** (SU-KMA) if Σ complies with Definition 3.6 where step 5 is replaced by

5. $\mathcal{A}$ succeeds if Verify$(pk, m, \sigma) =$ `True` and $(m, \sigma) \notin \{(m_1, \sigma_1), ..., (m_p, \sigma_p)\}$

Definition 3.7 (EU-CMA **Security)** *A signature scheme Σ is* **existentially unforgeable under a chosen-message attack** *(*EU-CMA*) if for all PPT adversaries $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following experiment is negligible as a function of k:*

1. KEYGEN(1^k) *is run to obtain a pair of keys (pk, sk)*
2. *$\mathcal{A}$ is given pk and allowed to interact with a signing oracle* SIGN$(sk, \cdot)$*, requesting signatures on as many messages as it likes. Let this attack be denoted as $\mathcal{A}_{pk}^{\text{SIGN}(\cdot)}$*
3. *Eventually, $\mathcal{A}$ outputs (m, σ)*
4. *$\mathcal{A}$ succeeds if* VERIFY$(pk, m, \sigma) =$ `True` *and* $m \notin \mathcal{M}$

The desired security notion for signature scheme is strong unforgeability under chosen message attack (SU-CMA). This notion is defined by the next security game:

Definition 3.8 (SU-CMA **Security Game and Advantage)** *The* SU-CMA *security game $G_{\Sigma}^{\text{SU-CMA}}$ is defined as a protocol between the challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

1. *$\mathcal{C}$ runs $(sk, pk) \leftarrow$* KEYGEN(λ) *and sends pk to $\mathcal{A}$*
2. *$\mathcal{A}$ adaptively chooses messages $m_1 \dots m_k$ and sends them to $\mathcal{C}$*
3. *$\mathcal{C}$ responds to each message with the signature $\sigma_i =$* SIGN(pk, sk, m)
4. *The adversary sends a message and a forgery (m^*, σ^*) to $\mathcal{C}$*
5. *$\mathcal{C}$ outputs*

$$\begin{cases} 1 & \text{if VERIFY}(pk, m^*, \sigma^*) = \texttt{True} \text{ and if } (m^*, \sigma^*) \neq (m_i, \sigma_i) \text{ for all } i = 1 \dots k \\ 0 & \text{otherwise} \end{cases}$$

The advantage of an SU-CMA *adversary $\mathcal{A}$ against the signature scheme Σ is defined as:*

$$\text{Adv}_{\Sigma}^{\text{SU-CMA}}(\mathcal{A}) = \Pr\left[\text{Exp}_{\Sigma}^{\text{SU-CMA}}(\mathcal{A}) = 1\right]$$

Definition 3.9 (SU-CMA **Security)** *A signature scheme Σ is said to be* SU-CMA *secure if for any adversary $\mathcal{A}$ that runs in probabilistic polynomial time (PPT) in the security parameter λ, the adversary's advantage $\text{Adv}_{\Sigma}^{\text{SU-CMA}}(\mathcal{A})$ is negligible in λ.*

Observations. CMA security can be attained from weaker primitives. Such constructions, exceeding the purpose of this thesis, are presented in detail in [YK05].

3.4 Elliptic Curve Cryptography

The first implicit appearance of elliptic curves is related to the work of Diophantus. The modern theory of elliptic curves started around 1930 with Hasse's research on the number of points on elliptic curves over finite fields [Has35]. This seminal work was later generalized by Weil's conjectures [And49].

Terminology: An Abelian group is a group satisfying the commutative law. For the definition of *genus* we refer the reader to [CFA+12].

Elliptic curves (see Definitions 3.10 and 3.11) are genus 1 algebraic curves. For the cryptographer, the very interesting property of an elliptic curve is that the set of points on the curve form an Abelian group. We recall two definitions for the elliptic curve notion, the second of which (Definition 3.11) is of interest to our purpose.

Definition 3.10 *An elliptic curve is a smooth projective curve of genus 1 with a distinguished point.*

Definition 3.11 *An elliptic curve* EC *over a finite prime field $\mathbb{F}_p$ of characteristic $p > 3$ can be described by its reduced Weierstraß form:*

$$E\colon y^2 = x^3 + ax + b \ . \tag{3.4}$$

We denote by $E(\mathbb{F}_p)$ the set of points $(x, y) \in \mathbb{F}_p^2$ satisfying equation (3.4), together with the point at infinity $\mathcal{O}$. $E(\mathbb{F}_p)$ is an additive Abelian group.

The leftmost part of Figure 3.7 shows an EC over the real numbers. An elliptic curve defined over a finite field is shown on the rightmost part of Figure 3.7.

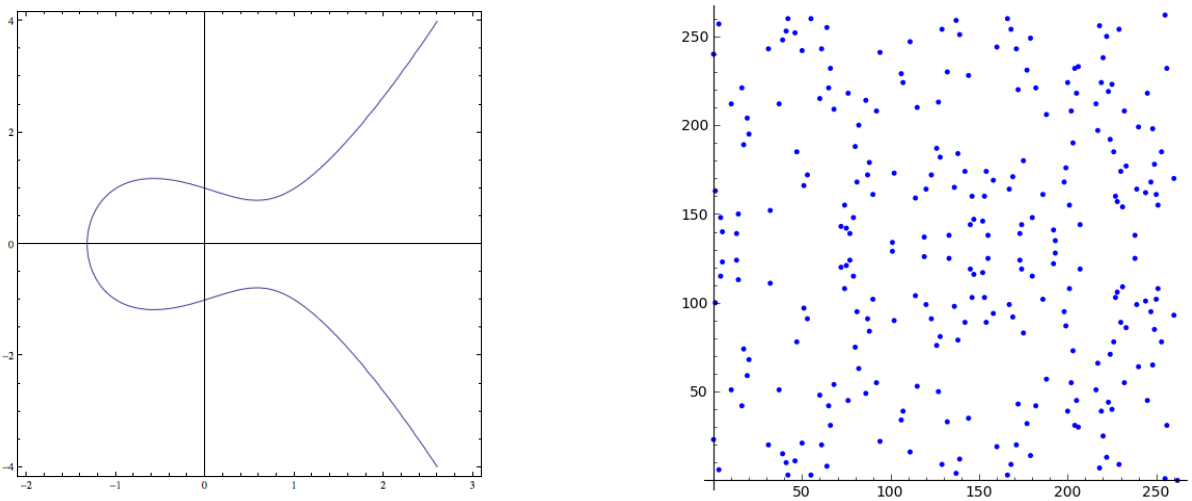


Figure 3.7: The EC $y^2 = x^3 - x + 1$ over $\mathbb{R}$ (leftmost image) and an EC defined over a finite field (rightmost image).

Elliptic Curve Cryptography (ECC)

Elliptic curves were first used for cryptographic purposes by Lenstra [Len87]. In 1984 Lenstra mentioned the idea of an elliptic curve factoring algorithm which was published by 1987 [Len87]. Seeing this as a breakthrough, Koblitz [Kob87] and V. Miller [Mil86] independently proposed to use the Discrete Logarithm Problem on an elliptic curve to define public-key cryptosystems (ECC).

ECCs rely on the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP, given P and $Q = [k]P$ compute k) or on the hardness of related problems such as ECDH or ECDDH [Bon98], which can be solved if ECDLP can be.

But choosing suitable curves was challenging. Moreover, random curves may prove weak. Recommendations from different standardisation bodies appeared. For example, NIST first provided their

Recommended Elliptic Curves for Federal Government Use in 1999 [oST99]. Elliptic curves over finite fields of the form $\mathbb{F}_{2^n}$ are of great interest for fast implementations as the arithmetic processors corresponding to such fields can be constructed easily.

The Elliptic Curve Diffie-Hellman (ECDH) and the Elliptic Curve Menezes-Qu-Vanstone (ECMQV) [LMQ+03] were proposed as key agreement and transport protocols and for the Elliptic Curve Digital Signature Algorithm (ECDSA) [JM99]. [JM99] was previously presented in Section 3.3.

ECC requires much smaller keys than those used in conventional public key cryptosystems, for an equal security level. The use of ECs therefore allows faster encryption, decryption and digital signatures.

Elliptic curves used in cryptography have the next properties:

– They are defined over a finite field $\mathbb{F}_q$, where q is either a prime or of the form 2^n
– The number of points on a cryptographic EC is ip, where p is prime and $i \in \{1, 2, 3, 4\}$
– p is typically a 192-bit integer

Hyperelliptic curves are a natural generalization of elliptic curves:

Hyperelliptic Curve Cryptography. Hyperelliptic curves are genus $g > 1$ curves. In constrast with well established ECCs whose security has been extensively analysed, hyperelliptic curve cryptography still includes many unknown parameters. We refer the reader to [CFA+12] for further details.

Pairing-Based Cryptography. Joux [Jou00] noticed that bilinear pairings on ECs have very important cryptographic applications. These applications were deemed so important that in 2013 Joux, Boneh and Franklin were awarded the Gödel Prize for this discovery. Initially used for cryptanalysis [MOV93], pairings became interesting for constructing new ID-based cryptographic primitives [BF01,Lys02,SOK00]. But the majority of these applications requires working with finite fields of prime power order.

Weil pairing was introduced by Weil in 1940 [Wei40] and represents a bilinear form on the points of order dividing n of an elliptic curve E.

Among the cryptographic applications of Weil pairing, we mention Joux's tri-partite Diffie–Hellman-like key exchanges [Jou00] and ID-based public key cryptosystems [BF01,Sma01] in which the public keys can be selected by the users.

The Tate pairing [Tat58, Tat63] is preferred in cryptographic applications for its relative computational efficiency.

In the next subsections, the basic properties and algorithms of importance for the arithmetic of elliptic curves will be overviewed: point addition (Algorithm 1), scalar multiplication (Algorithm 2) and Jacobian projective arithmetics (Algorithms 3.5 and 3.6). These are important for our results in Section 5.6.1.3.

Point Addition and Point Doubling

Algorithm 1 describes point addition on elliptic curves, and, implicitly, point doubling (when $P = Q$).

The inverse of point P is defined as $-P = (x_1, -y_1)$.

Jacobian Projective Arithmetics

To avoid modular inversions, implementers frequently work in the Jacobian projective coordinates system. The equation of an EC in the Jacobian projective coordinates system in the reduced Weierstraß form is:

$$E^{\mathcal{J}} : Y^2 = X^3 + aXZ^4 + bZ^6 .$$

The projective point (X, Y, Z) corresponds to the affine point $(X/Z^2, Y/Z^3)$. The point (X, Y, Z) is equivalent to any point (r^2X, r^3Y, rZ) with $r \in \mathbb{F}_p^*$.

We recall the addition (ECADD) and doubling (ECDBL) formulæ in the Jacobian projective coordinates system. Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ be two points of $E^{\mathcal{J}}(\mathbb{F}_p)$ with $P \neq \pm Q$.

Algorithm 1: Point Addition

Input: Two points $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \notin \{\mathcal{O}, -P\}$ on $E(\mathbb{F}_p)$.
Output: $R = (x_3, y_3) = P + Q$

if $P = Q$ **then**
  $\lambda \leftarrow \dfrac{3x_1^2 + a}{2y_1}$
end if
else
  $\lambda \leftarrow \dfrac{y_1 - y_2}{x_1 - x_2}$
end if
$x_3 \leftarrow \lambda^2 - x_1 - x_2$
$y_3 \leftarrow \lambda(x_1 - x_3) - y_1$
return $R := (x_3, y_3)$

$$\text{Algorithm ECDBL} = \begin{cases} S & \leftarrow & 4X_1Y_1^2 & \\ M & \leftarrow & 3X_1^2 + aZ_1^4 & \\ X_3 & \leftarrow & -2S + M^2 & \\ Y_3 & \leftarrow & -8Y_1^4 + M(S - X_3) & \\ Z_3 & \leftarrow & 2Y_1Z_1 & \\ P_3 & := & (X_3, Y_3, Z_3) & \text{return}(P_3 = 2P_1) \end{cases} \tag{3.5}$$

$$\text{Algorithm ECADD} = \begin{cases} U & \leftarrow & X_1Z_2^2 & \\ S & \leftarrow & Y_1Z_2^3 & \\ H & \leftarrow & X_2Z_1^2 - U & \\ R & \leftarrow & Y_2Z_1^3 - S & \\ X_3 & \leftarrow & -H^3 - 2UH^2 + R^2 & \\ Y_3 & \leftarrow & -SH^3 + R(UH^2 - X_3) & \\ Z_3 & \leftarrow & Z_1Z_2H & \\ P_3 & := & (X_3, Y_3, Z_3) & \text{return}(P_3 = P_1 + P_2) \end{cases} \tag{3.6}$$

Scalar Multiplication

In ECC, one has to compute scalar multiplications, *i.e.* compute $[k]P$, given P and an integer k. The Double-and-Add algorithm (Algorithm 2) is a way of doing so.

Algorithm 2: Double-and-Add

Input: A point P and an integer $k = (1, k_{N-2}, k_{N-3,} \ldots, k_0)_2$
Output: $[k]P$

$A \leftarrow P$
for $i = N - 2$ downto 0 **do**
  $A \leftarrow \text{ECDBL}(A)$
  if $k_i = 1$ **then**
    $A \leftarrow \text{ECADD}(A, P)$
  end if
end for
return A

CHAPTER 4

PROTOCOL DESIGN

Everything should be made as simple as possible, but not simpler.
Albert Einstein.

Summary

Designing secure and practical cryptographic protocols is a challenging task. This chapter focuses on reductionist security and includes the description of a co-signature protocol, a lightweight algorithm performing network authentication and an authenticated encryption scheme.

The core result of the thesis, detailed in Section 4.1, is a new form of contract co-signature, called *legal fairness*, that does not rely on third parties or arbitrators.

The proposed protocol is efficient, compact, fully distributed, fully dynamic, and provably secure in the Random Oracle Model. The protocol is illustrated for two parties using Schnorr's signature scheme.

Section 4.2 presents a lightweight algorithm allowing a verifier to collectively identify a community of provers. This protocol is more efficient than one-to-one node authentication, resulting in less communication, less computation, and hence a smaller overall energy consumption. The protocol is provably secure, and achieves zero-knowledge authentication of a time linear in the degree of the spanning tree.

The proposed authentication protocol may be adapted to better fit constraints: in the context of Internet of Things (IoT), communication is a very costly operation. We describe versions that reduce the amount of data sent by individual nodes, while maintaining security.

Section 4.3 introduces a novel mode of operation called OMD which uses a compression function for building a nonce-based authenticated encryption with associated data. OMD is instantiated with the specific compression functions of SHA-256 and SHA-512. OMD features higher quantitative security level, flexible key sizes and simpler operations than AES-GCM.

OMD is a second round candidate of the CAESAR competition. OMD was benchmarked both in software and hardware. Section 4.3.11 discusses further developments regarding authenticated encryption schemes.

4.1 Legally Fair Contract Signing without Keystones

When mutually distrustful parties wish to compute some joint function of their private inputs, they require a certain number of security properties to hold for that computation:
– *Privacy*: Nothing is learnt from the protocol besides the output;
– *Correctness*: The output is distributed according to the prescribed functionality;
– *Independence*: One party cannot make their inputs depend on the other parties' inputs;
– *Delivery*: An adversary cannot prevent the honest parties from successfully computing the functionality;
– *Fairness*: If one party receives output then so do all.
Any multi-party computation can be securely computed [Yao86, GMW87, Gol04, BOSW88, CCD88] as long as there is a honest majority [Lin08]. In the case where there is no such majority, and in particular in the two-party case, it is (in general[1]) impossible to achieve both fairness and guaranteed output delivery [Lin08, Cle86].

Weakening Fairness. To circumvent this limitation, several authors have put forth alternatives to fairness that try and capture the practical context (*e.g.* contract-signing, bank transactions, etc.). Three main directions have been explored:

1. *Gradual release models*: The output is not revealed all at once, but rather released gradually (*e.g.* bit per bit) so that, if an abort occurs, then the adversary has not learnt much more about the output than the honest party. This solution is unsatisfactory because it is expensive and may not work if the adversary is more computationally powerful [GHKL08, GAL91, Pin03, GDMPY06].
2. *Optimistic models*: A trusted server is setup but will not be contacted unless fairness is breached. The server is able to restore fairness afterwards, and this approach can be efficient, but the infrastructure requirements and the condition that the server be trusted limit the applicability of this solution [Mic03, ASW97, CC00]. In particular, the dispute-resolving third party must be endowed with functions beyond those usually required of a normal certification authority.
3. *Legally fair, or concurrent model*: The first party to receive output obtains an information dubbed the "keystone". The keystone by itself gives nothing and so if the first party aborts after receiving it, no damage has been done – if the second party aborts after receiving the result (say, a signature) then the first party is left with a useless keystone. But, as observed in [CKP04] for the signature to be enforced, it needs to be presented to a court of law, and legally fair signing protocols are designed so that this signature *and* the keystone give enough information to reconstruct the missing data. Therefore, if the cheating party wishes to enforce its signed contract in a court of law, it by doing so reveal the signature that the first party should receive, thereby restoring fairness [CKP04]. Legal fairness requires neither a trusted arbitrator nor a high degree of interaction between parties.

Lindell [Lin08] also introduces a notion of "legally enforceable fairness" that sits between legal fairness and optimistic models: a trusted authority may force a cheating party to act in some fashion, should their cheating be attested. In this case the keystone consists in a digitally signed cheque for an frighteningly high amount of money that the cheating party would have to pay if the protocol were to be aborted prematurely and the signature abused.

Concurrent Signatures. Chen *et al.* [CKP04] proposed a legally fair signature scheme based on ring signatures [RST01, AOS02] and designated verifier signatures [JSI96b], that is proven secure in the Random Oracle Model assuming the hardness of computing discrete logarithms.

Concurrent signatures rely on a property shared by ring and designated verifier signatures called "ambiguity". In the case of two-party ring signatures, one cannot say which of the two parties produced the signature – since either of two parties could have produced such an ambiguous signature, both parties can deny having produced it. However, within the ring, if A receives a signature then she knows that it is B who sent it. The idea is to put the ambiguity-lifting information in a "keystone". When that keystone is made public, both signatures become simultaneously binding.

Concurrent signatures schemes can achieve legal fairness depending on the context. However their construction is not *abuse-free* [BW00, GJDM99]: the party A holding the keystone can always determine

1. See [GHKL08] for a very specific case where completely fair two-party computation can be achieved.

whether to complete or abort the exchange of signatures, and can demonstrate this by showing an outside party the signature from B with the keystone, before revealing the keystone to B.

Our Results. We describe a new contract signing protocol that achieves legal fairness and abuse-freeness. This protocol is based on the well-known Schnorr signature protocol, and produces signatures *compatible* with standard Schnorr signatures. For this reason, and as we demonstrate, the new contract signing protocol is provably secure in the random oracle model under the hardness assumption of solving the discrete logarithm problem. Our construction can be adapted to other DLP schemes, such as most [2] of those enumerated in [HPM94], including Girault-Poupard-Stern [GPS06] and ElGamal [EG84].

4.1.1 Preliminaries

4.1.1.1 Schnorr Signatures

Schnorr digital signatures [Sch90] are an offspring of ElGamal [EG84] signatures. This family of signatures is obtained by converting interactive identification protocols (zero-knowledge proofs) into transferable proofs of interaction (signatures). This conversion process, implicitly used by ElGamal, was discovered by Feige, Fiat and Shamir [FFS88] and formalized by Abdalla, Bellare and Namprempre [AABN02].

We will refer to the original Schnorr signature protocol as "classical" Schnorr. This protocol consists in four algorithms:

- $\mathsf{Setup}(\ell)$: On input a security parameter ℓ, this algorithm selects large primes p, q such that $q \geq 2^\ell$ and $p - 1 \bmod q = 0$, as well as an element $g \in \mathbb{G}$ of order q in some multiplicative group $\mathbb{G}$ of order p, and a hash function $H : \{0,1\}^* \to \{0,1\}^\ell$. The output is a set of public parameters $\mathsf{pp} = (p, q, g, \mathbb{G}, H)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters, this algorithm chooses uniformly at random $x \xleftarrow{\$} \mathbb{Z}_q^\times$ and computes $y \leftarrow g^x$. The output is the couple $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} = x$ is kept private, and $\mathsf{pk} = y$ is made public.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: On input public parameters, a secret key, and a message m this algorithm selects a random $k \xleftarrow{\$} \mathbb{Z}_q^\times$, computes

$$\begin{aligned} r &\leftarrow g^k \\ e &\leftarrow H(m||r) \\ s &\leftarrow k - ex \bmod q \end{aligned}$$

  and outputs $\langle r, s \rangle$ as the signature of m.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: On input the public parameters, a public key, a message and a signature $\sigma = \langle r, s \rangle$, this algorithm computes $e \leftarrow H(m, r)$ and returns `True` if and only if $g^s y^e = r$; otherwise it returns `False`.

The security of classical Schnorr signatures was analysed by Pointcheval and Stern [PS96, PS00] using the Forking Lemma. Pointcheval and Stern's main idea is as follows: in the Random Oracle Model, the opponent can obtain from the forger two valid forgeries $\{\ell, s, e\}$ and $\{\ell, s', e'\}$ for the same oracle query $\{m, r\}$ but with different message digests $e \neq e'$. Consequently, $r = g^s y^{-e} = g^{s'} y^{-e'}$ and from that it becomes straightforward to compute the discrete logarithm of $y = g^x$. Indeed, the previous equation can be rewritten as $y^{e-e'} = g^{s'-s}$, and therefore:

$$y = g^{\frac{s'-s}{e-e'}} \;\Rightarrow\; \mathrm{Dlog}_g(y) = \frac{s'-s}{e-e'}$$

The Forking Lemma for Schnorr signatures is originally stated as follows:

Theorem 4.1 (Forking Lemma, [PS00]) *Let $\mathcal{A}$ be an attacker which performs within a time bound t_F an existential forgery under an adaptively chosen-message attack against the Schnorr signature, with probability ϵ_F. Assume that $\mathcal{A}$ makes q_h hashing queries to a random oracle and q_s queries to a signing oracle.*

Then there exists an adversary solving the discrete logarithm problem in subgroups of prime order in polynomial expected time. Assume that $\epsilon_F \geq 10(q_s + 1)(q_s + q_h)/q$, then the discrete logarithm problem in subgroups of prime order can be solved within expected time less that $120686\, q_h t_F / \epsilon_F$.

2. In a number of cases, *e.g.* DSA, the formulae of s do not lend themselves to security proofs.

This security reduction loses a factor $O(q_h)$ in the time-to-success ratio. Note that recent work by Seurin [Seu12] shows that this is essentially the best possible reduction to the discrete logarithm problem.

4.1.1.2 Concurrent Signatures

Let us give a more formal account of legal fairness as described in [CKP04, Lin08] in terms of concurrent signatures. Unlike classical contract-signing protocol, whereby contractors would exchange full-fledged signatures (*e.g.* [Gol83]), in a concurrent signature protocol there are "ambiguous" signatures that do not, as such, bind their author. This ambiguity can later be lifted by revealing some additional information: the "keystone". When the keystone is made public, both signatures become simultaneously binding.

Let $\mathcal{M}$ be a message space. Let $\mathcal{K}$ be the keystone space and $\mathcal{F}$ be the keystone fix space.

Definition 4.1 (Concurrent signature) *A concurrent signature is composed of the following algorithms:*

- $\mathsf{Setup}(k)$*: Takes a security parameter* k *as input and outputs the public keys* (y_A, y_B) *of all participants, a function* $\mathsf{KeyGen} : \mathcal{K} \to \mathcal{F}$*, and public parameters* pp *describing the choices of* $\mathcal{M}, \mathcal{K}, \mathcal{F}$ *and* KeyGen*.*
- $\mathsf{aSign}(y_i, y_j, x_i, h_2, M)$*: Takes as input the public keys* y_1 *and* y_2*, the private key* x_i *corresponding to* y_i*, an element* $h_2 \in \mathcal{F}$ *and some message* $M \in \mathcal{M}$*; and outputs an "ambiguous signature"*

$$\sigma = \langle s, h_1, h_2 \rangle$$

where $s \in \mathcal{S}, h_1, h_2 \in \mathcal{F}$*.*

- $\mathsf{aVerify}(\sigma, y_i, y_j, M)$*: Takes as input an ambiguous signature* $\sigma = \langle s, h_1, h_2 \rangle$*, public keys* y_i *and* y_j*, a message* M*; and outputs a boolean value, with the constraint that*

$$\mathsf{aVerify}\,(\sigma', y_j, y_i, M) = \mathsf{aVerify}\,(\sigma, y_i, y_j, M)$$

where $\sigma' = \langle s, h_2, h_1 \rangle$*.*

- $\mathsf{Verify}(k, \sigma, y_i, y_j, M)$*: Takes as input* $k \in \mathcal{K}$ *and* σ, y_i, y_j, M *as above; and checks whether* $\mathsf{KeyGen}(k) = h_2$*: If not it terminates with output* False*, otherwise it outputs the result of* $\mathsf{aVerify}(\sigma, y_i, y_j, M)$*.*

A valid concurrent signature is a tuple $\langle k, \sigma, y_i, y_j, M \rangle$ that is accepted by the Verify algorithm. Concurrent signatures are used by two parties A and B in the following way:

1. A and B run Setup to determine the public parameters of the scheme. We assume that A's public and private keys are y_A and x_A, and B's public and private keys are y_B and x_B.
2. Without loss of generality, we assume that A initiates the conversation. A picks a random keystone $k \in \mathcal{K}$, and computes $f = \mathsf{KeyGen}(k)$. A takes her own public key y_A and B's public key y_B and picks a message $M_A \in \mathcal{M}$ to sign. A then computes her ambiguous signature to be
$$\sigma_A = \langle s_A, h_A, f \rangle = \mathsf{aSign}(y_A, y_B, x_A, f, M_A).$$
3. Upon receiving A's ambiguous signature σ_A, B verifies the signature by checking that
$$\mathsf{aVerify}(s_A, h_A, f, y_A, y_B, M_A) = \mathsf{True}$$
If this equality does not hold, then B aborts. Otherwise B picks a message $M_B \in \mathcal{M}$ to sign and computes his ambiguous signature
$$\sigma_B = \langle s_B, h_B, f \rangle = \mathsf{aSign}(y_B, y_A, x_B, f, M_B)$$
then sends this back to A. Note that B uses the same value f in his signature as A did to produce σ_A.
4. Upon receiving B's signature σ_B, A verifies that
$$\mathsf{aVerify}(s_B, h_B, f, y_B, y_A, M_B) = \mathsf{True}$$
where f is the same keystone fix as A used in the previous steps. If the equality does not hold, then A aborts. Otherwise A sends keystone k to B.

At the end of this protocol, both $\langle k, \sigma_A \rangle$ and $\langle k, \sigma_B \rangle$ are binding, and accepted by the Verify algorithm.

Remark Note that A has an the upper hand in this protocol: Only when A releases the keystone do both signatures become simultaneously binding, and there is no guarantee that A will ever do so. Actually, since A controls the timing of the keystone release (if it is released at all), A may only reveal k to a third party C but withhold it from B, and gain some advantage by doing so. In other terms, concurrent signatures can be *abused* by A [BW00, GJDM99].

Chen *et al.* [CKP04] argue that there are situations where it is not in A's interest to try and cheat B, in which abuse-freeness is not necessary. One interesting scenario is credit card payment in the "four corner" model. Assume that B's signature is a payment to A. To obtain payment, A must channel *via* her acquiring bank C, which would communicate with B's issuing bank D. D would ensure that B receives both the signature and the keystone — as soon as this happens A is bound to her signature. Since in this scenario there is no possibility for A to keep B's signature private, fairness is eventually restored.

Example 4.1 *A concurrent signature scheme based on the ring signature algorithm of Abe et al. [AOS02] was proposed by Chen et al. [CKP04]:*

- Setup*: On input a security parameter ℓ, two large primes p and q are selected such that $q|p-1$. An element $g \in \mathbb{Z}_p^\times$ of order q is selected. The spaces $\mathcal{S} = \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} = \mathcal{K} = \{0,1\}^*$ are chosen. Two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_q$ are selected and we set* KeyGen $= H_1$*. Private keys x_A, x_B are selected uniformly at random from $\mathbb{Z}_q$ and the corresponding public keys are computed as $g^{x_i} \bmod p$.*
- aSign*: The algorithms takes as input y_i, y_j, x_i, h_2, M, verifies that $y_i \neq y_j$ (otherwise aborts), picks a random value $t \in \mathbb{Z}_q$ and computes*

$$\begin{aligned} h &= H_2\left(g^t y_j^{h_2} \bmod p \| M\right) \\ h_1 &= h - h_2 \bmod q \\ s &= t - h_1 x_i \bmod q \end{aligned}$$

  where $\|$ denotes concatenation. The algorithm outputs $\langle s, h_1, h_2 \rangle$.
- aVerify*: This algorithm takes as input s, h_1, h_2, y_i, y_j, M and checks whether the following equation holds:*

$$h_1 + h_2 = H_2\left(g^s y_i^{h_1} y_j^{h_2} \bmod p \| M\right) \bmod q$$

The security of this scheme can be proven in the Random Oracle model assuming the hardness of computing discrete logarithms in $\mathbb{Z}_p^\times$.

4.1.1.3 Legal Fairness for Concurrent Signatures

A concurrent signature scheme is secure when it achieves existential unforgeability, ambiguity and fairness against an active adversary that has access to a signature oracle. We define these notions in terms of games played between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In all security games, $\mathcal{A}$ can perform any number of the following queries:

- KeyGen queries: $\mathcal{A}$ can receive a keystone fix $f =$ KeyGen(k) where k is chosen by the challenger [3].
- KeyReveal queries: $\mathcal{A}$ can request that $\mathcal{C}$ reveals which k was chosen to produce a keystone fix f in a previous KeyGen query. If f was not a previous KeyGen query output then $\mathcal{C}$ returns $\perp$.
- aSign queries: $\mathcal{A}$ can request an ambiguous signature for any message of his choosing and any pair of users [4].
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.

Definition 4.2 (Unforgeability) *The notion of existential unforgeability for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Finally, $\mathcal{A}$ outputs a tuple $\sigma = \langle s, h_1, f \rangle$ where $s \in \mathcal{S}, h_1, f \in \mathcal{F}$, along with public keys y_C, y_D and a message $M \in \mathcal{M}$.*

$\mathcal{A}$ wins the game if aVerify *accepts σ and either of the following holds:*

3. The algorithm KeyGen being public, $\mathcal{A}$ can compute KeyGen(k) for any k of her choosing.
4. Note that with this information and using KeyGen queries, $\mathcal{A}$ can obtain concurrent signatures for any message and any user pair.

– *$\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query aSign on (y_C, y_D, f, M) nor on (y_D, y_C, h_1, M).*
– *$\mathcal{A}$ did not query aSign on (y_C, y_i, f, M) for any $y_i \neq y_C$, and did not query SKExtract on y_C, and f is the output of* KeyGen*: either an answer to a KeyGen query, or $\mathcal{A}$ can produce a k such that $k = \mathsf{KeyGen}(k)$.*

The last constraint in the unforgeability security game corresponds to the situation where $\mathcal{A}$ knows one of the private keys (as is the case if $\mathcal{A} = A$ or B).

Definition 4.3 (Ambiguity) *The notion of ambiguity for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to $\mathcal{A}$.*
2. *Phase 1: $\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Challenge: $\mathcal{A}$ selects a challenge tuple (y_i, y_j, M) where y_i, y_j are public keys and $M \in \mathcal{M}$. In response, $\mathcal{C}$ selects a random $k \in \mathcal{K}$, a random $b \in \{0, 1\}$ and computes $f = \mathsf{KeyGen}(k)$. If $b = 0$, then $\mathcal{C}$ outputs*

$$\sigma_1 = \langle s_1, h_1, f \rangle = \mathsf{aSign}(y_i, y_j, x_i, f, M)$$

Otherwise, if $b = 1$ then $\mathcal{C}$ computes

$$\sigma_2 = \langle s_2, h_2, f \rangle = \mathsf{aSign}(y_j, y_i, x_i, f, M)$$

but outputs $\sigma_2' = \langle s_2, f, h_2 \rangle$ instead.
4. *Phase 2: $\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
5. *Finally, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$.*

$\mathcal{A}$ wins the game if $b = b'$ and if $\mathcal{A}$ made no KeyReveal query on f, h_1 or h_2.

Definition 4.4 (Fairness) *The notion of fairness for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Finally, $\mathcal{A}$ chooses two public keys y_C, y_D and outputs $k \in \mathcal{K}$ and $S = (s, h_1, f, y_C, y_D, M)$ where $s \in \mathcal{S}$, $h_1, f \in \mathcal{F}$, $M \in \mathcal{M}$.*

$\mathcal{A}$ wins the game if $\mathsf{aVerify}(S)$ *accepts and either of the following holds:*
– *f was output from a KeyGen query, no KeyReveal query was made on f, and* Verify *accepts $\langle k, S \rangle$.*
– *$\mathcal{A}$ can output $S' = (s', h_1', f, y_D, y_C, M')$ where* $\mathsf{aVerify}(S')$ *accepts and* $\mathsf{Verify}(k, S)$ *accepts, but* $\mathsf{Verify}(k, S')$ *rejects.*

This definition of fairness formalizes the idea that B cannot be left in a position where a keystone binds his signature to him while A's initial signature is not also bound to A. It does not, however, guarantee that B will ever receive the necessary keystone.

4.1.2 Legally Fair Co-Signatures

4.1.2.1 Legal Fairness Without Keystones

The main idea builds on the following observation: Every signature exchange protocol is plagued by the possibility that the last step of the protocol is not performed. Indeed, it is in the interest of a malicious party to get the other party's signature without revealing its own. As a result, the best one can hope for is that a trusted third party can eventually restore fairness.

To avoid this destiny, the proposed protocol does *not* proceed by sending A's signature to B and vice versa. Instead, we construct a *joint signature*, or *co-signature*, of both A and B. By design, there are no signatures to steal — and stopping the protocol early does not give the stopper a decisive advantage. More precisely, the contract they have agreed upon is the best thing an attacker can gather, and if she ever wishes to enforce this contract by presenting it to a court of law, she would confirm her own commitment to it as well as the other party's. Therefore, if one can construct co-signatures without intermediary individual signatures being sent, legal fairness can be achieved without keystones.

Since keystones can be used by the party having them to abuse the other [CKP04], the co-signature setting provides an interesting alternative to concurrent signatures.

Schnorr Co-signatures. To illustrate the new idea, we now discuss a legally fair contract-signing protocol built from the well-known Schnorr signature protocol, that produces signatures *compatible* with standard Schnorr signatures. This contract signing protocol is provably secure in the random oracle model under the hardness assumption of solving the discrete logarithm problem.

The construction can be adapted to other DLP schemes, such as most [5] of those enumerated in [HPM94], including Girault-Poupard-Stern [GPS06] and ElGamal [EG84].

Our proposal does not consider methods for dealing with revocation of keys at different levels separately.

– Setup: An independent (not necessarily trusted) authority generates a classical Schnorr parameter-set p, q, g which is given to A and B. Each user U generates a usual Schnorr public key $y_U = g^{x_U}$ and publishes y_U on a public directory $\mathcal{D}$ (see Figure 4.1). To determine the co-signature public-key $y_{A,B}$ of the pair $\langle A, B\rangle$, a verifier consults $\mathcal{D}$ and simply computes $y_{A,B} = y_A \times y_B$. Naturally, $y_{A,B} = y_{B,A}$.

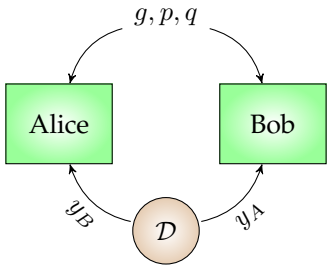


Figure 4.1: Public directory $\mathcal{D}$ distributing the public keys.

– Cosign: To co-sign a message m, A and B compute a common r and a common s, one after the other. Without loss of generality we assume that B initiates the co-signature.
  – During the first phase (Figure 4.2), B chooses a private random number k_B and computes $r_B \leftarrow g^{k_B}$. He commits to that value by sending to A a message digest $\rho \leftarrow H(0\|r_B)$. A chooses a private random number k_A, computes $r_A \leftarrow g^{k_A}$ and sends r_A to B. B replies with r_B, which A checks against the earlier commitment ρ. Both parties compute $r \leftarrow r_A \times r_B$, and $e \leftarrow H(1\|m\|r)$, where m is the message to be co-signed.
  – During the second phase of the protocol, B sends $s_B \leftarrow k_B - e \times x_B \bmod q$ to A. A replies with $s_A \leftarrow k_A - e \times x_A \bmod q$. Both users compute $s \leftarrow s_A + s_B \bmod q$.

Alice		*Bob*
$y_{A,B} \leftarrow y_A \times y_B$		$y_{A,B} \leftarrow y_A \times y_B$
$k_A \xleftarrow{\$} \mathbb{Z}_q^*$		$k_B \xleftarrow{\$} \mathbb{Z}_q^*$
$r_A \leftarrow g^{k_A}$		$r_B \leftarrow g^{k_B}$
		$\rho \leftarrow H(0\|r_B)$
	$\xleftarrow{\rho}$	
	$\xrightarrow{r_A}$	
	$\xleftarrow{r_B}$	
if $H(0\|r_B) \neq \rho$ **abort**		
$r \leftarrow r_A \times r_B$		$r \leftarrow r_A \times r_B$
$e \leftarrow H(1\|m\|r)$		$e \leftarrow H(1\|m\|r)$
$s_A \leftarrow k_A - ex_A \bmod q$		$s_B \leftarrow k_B - ex_B \bmod q$
	$\xleftarrow{s_B}$	
	$\xrightarrow{s_A}$	
$s \leftarrow s_A + s_B \bmod q$		$s \leftarrow s_A + s_B \bmod q$

Figure 4.2: Generating the Schnorr co-signature of message m.

5. In a number of cases, *e.g.* DSA, the formulae of s do not lend themselves to security proofs.

– Verify: As in the classical Schnorr signature, the co-signature $\{r, s\}$ is checked for a message m by computing $e \leftarrow H(m\|r)$, and checking whether $g^s y^e = r$ (Figure 4.3). If the equality holds, then the co-signature binds both A and B to m; otherwise neither party is tied to m.

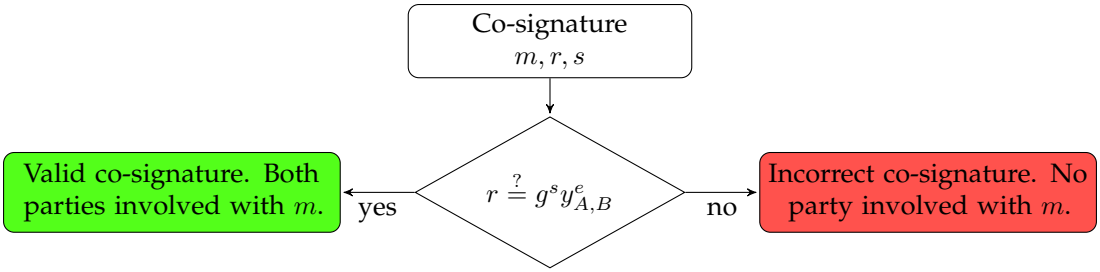


Figure 4.3: Verification of a Schnorr co-signature m, r, s.

Remark Note that during the co-signature protocol, A might decide not to respond to B: In that case, A would be the only one to have the complete co-signature. This is a breach of fairness insofar as A can benefit from the co-signature and not B, but the protocol is abuse-free: A cannot use the co-signature as a proof that B, and B alone, committed to m. Furthermore, it is not a breach of legal fairness: If A presents the co-signature in a court of law, she *ipso facto* reveals her commitment as well.

Remark In a general fair-contract signing protocol, A and B can sign different messages m_A and m_B. Using the co-signature construction requires that A and B agree first on the content of a single message m.

Security Analysis The security of the co-signature scheme essentially builds on the unforgeability of classical Schnorr signatures. Since there is only one co-signature output, the notion of ambiguity does not apply *per se* — albeit we will come back to that point later on. The notion of fairness is structural in the fact that a co-signature, as soon as it is binding, is binding for *both* parties.

As for concurrent signatures, an adversary $\mathcal{A}$ has access to an unlimited amount of conversations and valid co-signatures, *i.e.* $\mathcal{A}$ can perform the following queries:
– Hash queries: $\mathcal{A}$ can request the value of $H(x)$ for a x of its choosing.
– CoSign queries: $\mathcal{A}$ can request a valid co-signature r, s for a message m and a public key $y_{C,D}$ of its choosing.
– Transcript queries: $\mathcal{A}$ can request a valid transcript $(\rho, r_C, r_D, s_C, s_D)$ of the co-signing protocol for a message m of its choosing, between users C and D of its choosing.
– SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.
– Directory queries: $\mathcal{A}$ can request the public key of any user U.
The following definition captures the notion of unforgeability in the co-signing context:

Definition 4.5 (Unforgeability) *The notion of unforgeability for co-signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:*

1. *The* Setup *algorithm is run and all public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Finally, $\mathcal{A}$ outputs a tuple $(m, r, s, y_{C,D})$.*

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, r, s) = \mathsf{True}$ *and there exist public keys $y_C, y_D \in \mathcal{D}$ such that $y_{C,D} = y_C y_D$ and either of the following holds:*
– *$\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query CoSign on $m, y_{C,D}$, and did not query Transcript on m, y_C, y_D nor m, y_D, y_C.*
– *$\mathcal{A}$ did not query Transcript on m, y_C, y_i for any $y_i \neq y_C$ and did not query SKExtract on y_C, and did not query CoSign on m, y_C, y_i for any $y_i \neq y_C$.*
We shall say that a co-signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.

To prove that the Schnorr-based scheme described above is secure we use the following strategy: Assuming an efficient forger $\mathcal{A}$ for the co-signature scheme, we turn it into an efficient forger $\mathcal{B}$ for Schnorr signatures, then invoke the Forking Lemma to prove the existence of an efficient solver $\mathcal{C}$ for the discrete logarithm problem. All proofs hold in the Random Oracle model.

Since the co-signing protocol gives the upper hand to the last-but-one speaker there is an asymmetry: Alice has more information than Bob. Therefore we address two scenarios: When the attacker plays Alice's role, and when the attacker plays Bob's.

Adversary Attacks Bob.

Theorem 4.2 *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}_{Alice}$ plays the role of Alice and is able to forge in polynomial time a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}_{Alice}$ can break that DLP instance with high probability in polynomial time.*

Proof: The proof consists in constructing a simulator $\mathcal{S}_{\text{Bob}}$ that interacts with the adversary and forces it to actually produce a classical Schnorr forgery. Here is how this simulator behaves at each step of the protocol.

1. *Key Establishment Phase:*

   $\mathcal{S}_{\text{Bob}}$ is given a target DLP instance $\{y, g, p, q\}$. As a simulator, $\mathcal{S}_{\text{Bob}}$ emulates not only Bob, but also all oracles and the directory $\mathcal{D}$ (see Figure 4.4).

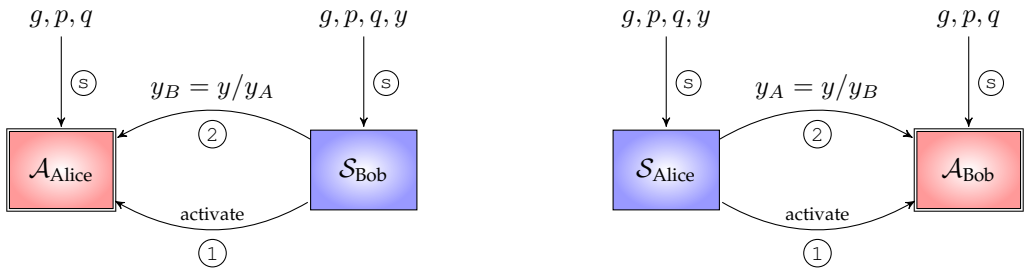


Figure 4.4: The simulator $\mathcal{S}_{\text{Bob}}$ (left) or $\mathcal{S}_{\text{Alice}}$ (right) answers the attacker's queries to the public directory $\mathcal{D}$.

   $\mathcal{S}_{\text{Bob}}$ injects the target y into the game, namely by posting in the directory the "public-key" $y_B \leftarrow y \times y_A^{-1}$.

   To inject a target DLP instance $y \leftarrow g^x$ into $\mathcal{A}$, the simulator $\mathcal{S}_{\text{Bob}}$ reads $y_{\mathcal{A}}$ from the public directory and poses as an entity whose public-key is $y_{\mathcal{S}} \leftarrow y \times y_{\mathcal{A}}^{-1}$. It follows that $y_{\mathcal{A},\mathcal{S}}$, the common public-key of $\mathcal{A}$ and $\mathcal{S}$ will be precisely $y_{\mathcal{A},\mathcal{S}} \leftarrow y_{\mathcal{S}} \times y_{\mathcal{A}}$ which, by construction, is exactly y.

   Then $\mathcal{S}_{\text{Bob}}$ activates $\mathcal{A}_{\text{Alice}}$, who queries the directory and gets y_B. At this point in time, $\mathcal{A}_{\text{Alice}}$ is tricked into believing that she has successfully established a common co-signature public-key set $\{g, p, q, y\}$ with the "co-signer" $\mathcal{S}_{\text{Bob}}$.

2. *Query Phase:*

   $\mathcal{A}_{\text{Alice}}$ will now start to present queries to $\mathcal{S}_{\text{Bob}}$. In a "normal" attack, $\mathcal{A}_{\text{Alice}}$ and Bob would communicate with a random oracle $\mathcal{O}$ representing the hash function H. However, here, the simulator $\mathcal{S}_{\text{Bob}}$ will play $\mathcal{O}$'s role and answer $\mathcal{A}_{\text{Alice}}$'s hashing queries.

   $\mathcal{S}_{\text{Bob}}$ must respond to three types of queries: *hashing queries*, *co-signature queries* and *transcript queries*. $\mathcal{S}_{\text{Bob}}$ will maintain an oracle table T containing all the hashing queries performed throughout the attack. At start $T \leftarrow \varnothing$. When $\mathcal{A}_{\text{Alice}}$ submits a hashing query q_i to $\mathcal{S}_{\text{Bob}}$, $\mathcal{S}_{\text{Bob}}$ answers as shown in Algorithm 3.

   When $\mathcal{A}_{\text{Alice}}$ submits a co-signature query to $\mathcal{S}_{\text{Bob}}$, $\mathcal{S}_{\text{Bob}}$ proceeds as explained in Algorithm 4.

   Finally, when $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending $\{m, \rho, r_A, r_B, s_B, s_A\}$ from a previously successful interaction.

3. *Output Phase:*

   After performing queries, $\mathcal{A}_{\text{Alice}}$ eventually outputs a co-signature m, r, s valid for $y_{\mathcal{A},\mathcal{S}}$ where $r = r_A r_B$ and $s = s_A + s_B$. By design, these parameters are those of a classical Schnorr signature and therefore $\mathcal{A}_{\text{Alice}}$ has produced a classical Schnorr forgery.

Algorithm 3: Hashing oracle simulation.

Input: A hashing query q_i from $\mathcal{A}$

if $\exists e_i, \{q_i, e_i\} \in T$ **then**
  $\rho \leftarrow e_i$
  else
    $\rho \xleftarrow{\$} \mathbb{Z}_q^{\times}$
  end if
  Append $\{q_i, \rho\}$ to T
end if
return ρ

Algorithm 4: Co-signing oracle simulation.

Input: A co-signature query m from $\mathcal{A}_{\text{Alice}}$

$s_B, e \xleftarrow{\$} \mathbb{Z}_q^*$
$r_B \leftarrow g^{s_B} y^e$
Send $H(0 \| r_B)$ to $\mathcal{A}_{\text{Alice}}$
Receive r_A from $\mathcal{A}_{\text{Alice}}$
$r \leftarrow r_A \times r_B$
$u \leftarrow 1 \| m \| r$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
  abort
  else
    Append $\{u, e\}$ to T
  end if
end if
return s_B

To understand $\mathcal{S}_{\text{Bob}}$'s co-signature reply (Algorithm 4), assume that $\mathcal{A}_{\text{Alice}}$ is an honest Alice who plays by the protocol's rules. For such an Alice, $\{s, r\}$ is a valid signature with respect to the common co-signature public-key set $\{g, p, q, y\}$. There is a case in which $\mathcal{S}_{\text{Bob}}$ aborts the protocol before completion: this happens when it turns out that $r_A \times r_B$ has been previously queried by $\mathcal{A}_{\text{Alice}}$. In that case, it is no longer possible for $\mathcal{S}_{\text{Bob}}$ to reprogram the oracle, which is why $\mathcal{S}_{\text{Bob}}$ must abort. Since $\mathcal{A}_{\text{Alice}}$ does not know the random value of r_B, such a bad event would only occur with a negligible probability exactly equal to q_h/q (where q_h is the number of queries to the hashing oracle).

Therefore, $\mathcal{A}_{\text{Alice}}$ is turned into a forger for the target Schnorr instance with probability $1 - q_h/q$. Since $\mathcal{A}_{\text{Alice}}$ succeeds with probability ϵ_F, $\mathcal{A}_{\text{Alice}}$'s existence implies the existence of a Schnorr signature forger of probability $\epsilon_S = (1 - q_h/q)\epsilon_F$, which by the Forking Lemma shows that there exists a polynomial adversary breaking the chosen DLP instance with high probability.

□

Being an attacker, at some point $\mathcal{A}_{\text{Alice}}$ will output a forgery $\{m', r', s'\}$. From here on we use the Forking Lemma and transform $\mathcal{A}_{\text{Alice}}$ into a DLP solver as described by Pointcheval and Stern in [PS00, Theorem 14].

Adversary Attacks Alice. The case where $\mathcal{A}$ targets A is similar but somewhat simpler, and the proof follows the same strategy.

Theorem 4.3 *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}_{Bob}$ plays the role of Bob and is able to forge a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}_{Bob}$ can break that DLP instance in polynomial time with high probability.*

Proof: Here also the proof consists in constructing a simulator, $\mathcal{S}_{\text{Alice}}$, that interacts with the adversary and forces it to actually produce a classical Schnorr forgery. The simulator's behaviour at different stages of the security game is as follows:

1. *The Key Establishment Phase:*

   $\mathcal{S}_{\text{Alice}}$ is given a target DLP instance $\{y, g, p, q\}$. Again, $\mathcal{S}_{\text{Alice}}$ impersonates not only Alice, but also $\mathcal{O}$ and $\mathcal{D}$. $\mathcal{S}_{\text{Alice}}$ injects the target y into the game as described in Section 4.1.2.1. Now $\mathcal{S}_{\text{Alice}}$ activates $\mathcal{A}_{\text{Bob}}$, who queries $\mathcal{D}$ (actually controlled by $\mathcal{S}_{\text{Alice}}$) to get y_B. $\mathcal{A}_{\text{Bob}}$ is thus tricked into believing that it has successfully established a common co-signature public-key set $\{g, p, q, y\}$ with the "co-signer" $\mathcal{S}_{\text{Alice}}$.

2. *The Query Phase:*

   $\mathcal{A}_{\text{Bob}}$ will now start to present queries to $\mathcal{S}_{\text{Alice}}$. Here as well, $\mathcal{S}_{\text{Alice}}$ will play $\mathcal{O}$'s role and will answer $\mathcal{A}_{\text{Bob}}$'s hashing queries.

   Again, $\mathcal{S}_{\text{Alice}}$ must respond to hashing queries and co-signature queries. Hashing queries are answered as shown in Algorithm 3. When $\mathcal{A}_{\text{Bob}}$ submits a co-signature query to $\mathcal{S}_{\text{Alice}}$, $\mathcal{S}_{\text{Alice}}$ proceeds as explained in Algorithm 5.

Algorithm 5: Co-signing oracle simulation for $\mathcal{S}_{\text{Alice}}$.

Input: A co-signature query m from $\mathcal{A}_{\text{Bob}}$

Receive ρ from $\mathcal{A}_{\text{Bob}}$
Query T to retrieve r_B such that $H(0\|r_B) = \rho$
$e, s_A \xleftarrow{\$} \mathbb{Z}_q$
$r \leftarrow r_B g^{s_A} y^e$
$u \leftarrow 1\|m\|r$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
    abort
    else
        Append $\{u, e\}$ to T
    end if
end if
$r_A \leftarrow r \times r_B^{-1}$
Send r_A to $\mathcal{A}_{\text{Bob}}$
Receive r_B from $\mathcal{A}_{\text{Bob}}$; this r_B is not used by $\mathcal{S}_{\text{Alice}}$
Receive s_B from $\mathcal{A}_{\text{Bob}}$
return s_A

   $\mathcal{S}_{\text{Alice}}$ controls the oracle $\mathcal{O}$, and as such knows what is the value of r_B that $\mathcal{A}_{\text{Bob}}$ is committed to. The simulator is designed to trick $\mathcal{A}_{\text{Bob}}$ into believing that this is a real interaction with Alice, but Alice's private key is not used.

3. *Output*:

   Eventually, $\mathcal{A}_{\text{Bob}}$ produces a forgery that is a classical Schnorr forgery $\{m, r, s\}$.

Algorithm 5 may fail with probability $1/q$. Using the Forking Lemma again, we transform $\mathcal{A}_{\text{Bob}}$ into an efficient solver of the chosen DLP instance. □

4.1.2.2 Concurrent Co-signatures

4.1.2.2.1 Proofs of Involvement. We now address a subtle weakness in the protocol described in the previous section, which is not captured by the fairness property *per se* and that we refer to as the existence of "proofs of involvement". Such proofs are not valid co-signatures, and would not normally be accepted by verifiers, but they nevertheless are valid evidence establishing that one party committed to a message. In a legally fair context, it may happen that such evidence is enough for one party to win a trial against the other — who lacks both the co-signature, and a proof of involvement.

Example 4.2 *In the co-signature protocol of Figure 4.2, s_B is not a valid Schnorr signature for Bob. Indeed, we have $g^{s_B} y_B^e = r_B \neq r$. However, Alice can construct $s' = s_B - k_A$, so that m, r, s' forms a valid classical signature of Bob* alone *on m.*

Example 4.2 illustrates the possibility that an adversary, while unable to forge a co-signature, may instead use the information to build a valid (mono-) signature. Note that Alice may opt for a weaker proof of involvement, for instance by demonstrating her possession of a valid signature using any zero-knowledge protocol.

A straightforward patch is to refrain from using the public keys y_A, y_B for both signature and co-signature — so that attempts at constructing proofs of involvement become vain. For instance, every user could have a key $y_U^{(1)}$ used for classical signature and for certifying a key $y_U^{(2)}$ used for co-signature[6]. If an adversary generates a classical signature from a co-signature transcript as in Example 4.2, she actually reveals her harmful intentions.

However, while this exposes the forgery — so that honest verifiers would reject such a signature — the perpetrator remains anonymous. There are scenarios in which this is not desirable, *e.g.* because it still proves that B agreed (with some unknown and dishonest partner) on m.

Note that the existence of proof of involvement is not necessary and depends on the precise choice of underlying signature scheme.

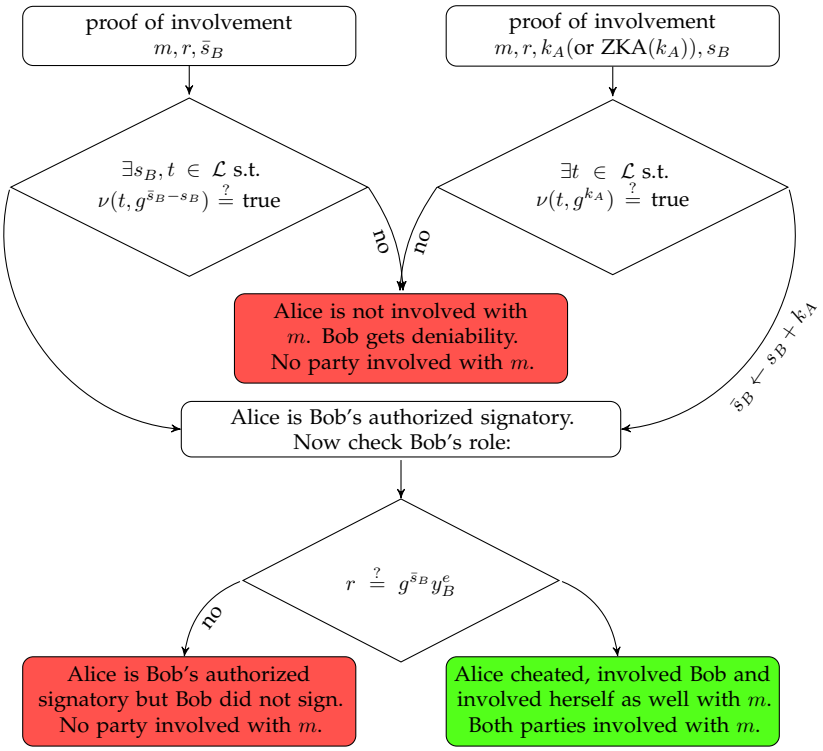


Figure 4.5: The verification procedure: **proof of involvement**.

4.1.2.3 Concurrent Co-signatures

In the interest of fairness, the best we can ask is that if A tries to incriminate B on a message they both agreed upon, she cannot do so anonymously.

To enforce fairness on the co-signature protocol, we ask that the equivalent of a keystone is transmitted first; so that in case of dispute, the aggrieved party has a legal recourse. First we define the notion of an authorized signatory credential:

6. The key $y_U^{(2)}$ may be derived from $y_U^{(1)}$ in some way, so that the storage needs of $\mathcal{D}$ are the same as for classical Schnorr.

Definition 4.6 (Authorized signatory credential) *The data field*

$$\Gamma_{Alice,Bob} = \{Alice, Bob, k_A, \sigma(g^{k_A} \| Alice \| Bob)\}$$

is called an authorized signatory credential *given by Alice to Bob, where σ is some publicly known auxiliary signature algorithm.*

Any party who gets $\Gamma_{\text{Alice,Bob}}$ can check its validity, and releasing $\Gamma_{\text{Alice,Bob}}$ is *by convention* functionally equivalent to Alice giving her private key x_A to Bob. A valid signature by Bob on a message m exhibited with a valid $\Gamma_{\text{Alice,Bob}}$ is *legally* defined as encompassing the meaning ($\Rrightarrow$) of Alice's signature on m:

$$\{\Gamma_{\text{Alice,Bob}}, \text{signature by Bob on } m\} \Rrightarrow \text{signature by Alice on } m$$

Second, the co-signature protocol of Figure 4.2 is modified by requesting that Alice provide t to Bob. Bob stores this in a local memory $\mathcal{L}$ along with s_B. Together, t and s_B act as a keystone enabling Bob (or a verifier, *e.g.* a court of law) to reconstruct $\Gamma_{\text{Alice,Bob}}$ if Alice exhibits a (fraudulent) signature binding Bob alone with his co-signing public key.

Therefore, should Alice try to exhibit as in Example 4.2 a signature of Bob alone on a message they both agreed upon (which is known as a fraud), the court would be able to identify Alice as the fraudster.

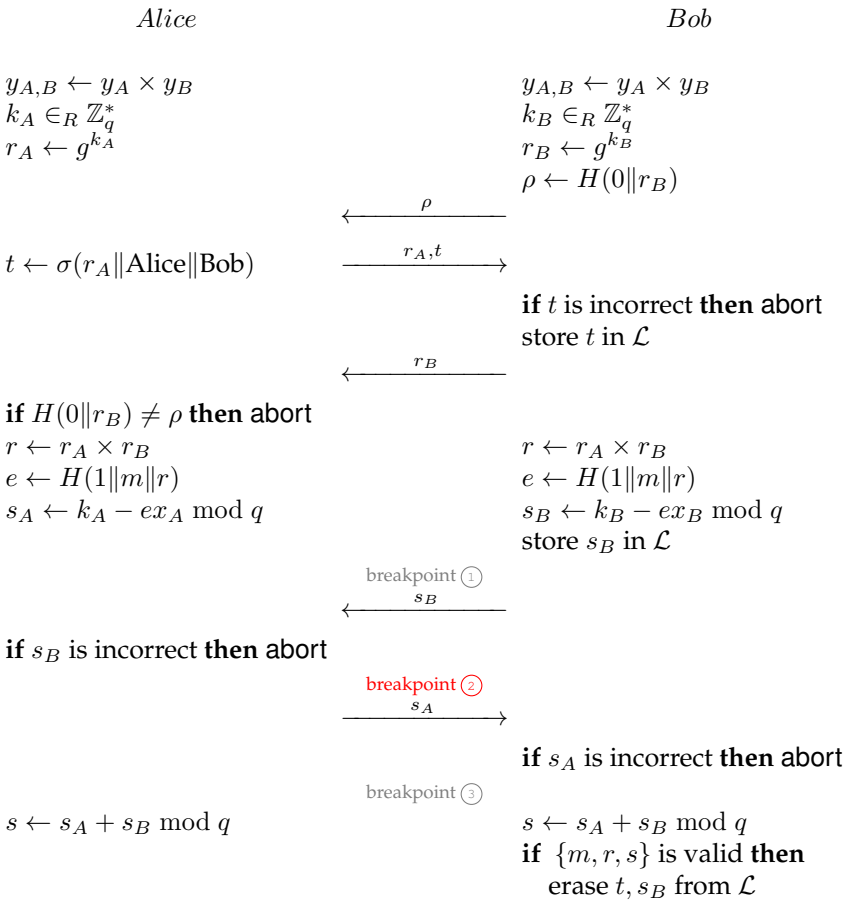


Figure 4.6: The legally fair co-signature of message m.

The modified signature protocol is described in Figure 4.6. Alice has only one window of opportunity to try and construct a fraudulent signature of Bob: by stopping the protocol at breakpoint (2) and using the information s_B [7].

Indeed, if the protocol is interrupted before breakpoint (1), then no information involving m was released

7. If Bob transmits a wrong or incorrect s_B, this will be immediately detected by Alice as $r_B \neq g^{s_B} y_B^e$. Naturally, in such a case, Bob never sent any information binding him to the contract anyway.

by any of the parties: The protocol's trace can be simulated without Bob's help as follows [8]

$$\begin{aligned} s_B, r &\xleftarrow{\$} \mathbb{Z}_q \\ e &\leftarrow H(1\|m\|r\|\text{Alice}\|\text{Bob}) \\ r_B &\leftarrow g^{s_B} y_B^e \\ r_A &\leftarrow r \times r_B^{-1} \\ t &\leftarrow \sigma(r_A\|\text{Alice}\|\text{Bob}) \\ \rho &\leftarrow H(0\|r_B) \end{aligned}$$

and Bob has only received from Alice the signature of a random integer.

If Alice and Bob successfully passed the normal completion breakpoint (3), *both* parties have the co-signature, and are provably committed to m.

8. No time-out mechanism has been taken into account.

4.2 Multi-Party Authentication Protocols

A growing market focuses on lightweight devices, whose low cost and easy production allow for creative and pervasive uses. The Internet of Things (IoT) consists in spatially distributed nodes that form a network, able to control or monitor physical or environmental conditions (such as temperature, pressure, image and sound), perform computations or store data. IoT nodes are typically low-cost devices with limited computational resources and limited battery. They transmit the data they acquire through the network to a gateway, also called the transceiver, which collects information and sends it to a processing unit. Nodes are usually deployed in hostile environments, and are therefore susceptible to physical attacks, harsh weather conditions and communication interferences.

Due to the open and distributed nature of the IoT, security is key to the entire network's proper operation [VPH+11]. However, the lightweight nature of sensor nodes heavily restricts the type of cryptographic operations that they can perform, and the constrained power resources make any communication costly.

This section describes an authentication protocol that establishes network integrity, and leverages the distributed nature of computing nodes to alleviate individual computational effort. This enables the base station to identify which nodes need replacement or attention.

This is most useful in the context of wireless sensor networks and the IoT, but applies equally well to mesh network authentication and similar situations.

Related Work: Zero Knowledge (ZK) protocols have been considered for authentication of wireless sensor networks. For instance, Anshul and Roy [AR05] describe a modified version of the Guillou-Quisquater identification scheme [GQ88], combined with the μTesla protocol [PST+02] for authentication broadcast in constrained environments. We stress that the purpose of the scheme of [AR05], and similar ones, is to authenticate the base station.

Aggregate signature schemes such as [BGLS03, ZQWZ10] may be used to achieve the goal pursued here – however they are intrinsically non-interactive, and the most efficient aggregate constructions use elliptic curve pairings, which require powerful devices.

Closer to our concerns, [UMS11] describes a ZK network authentication protocol, but it only authenticates two nodes at a time, and the base station acts like a trusted third party. As such it takes a very large number of interactions to authenticate the network as a whole.

What we propose instead is a collective perspective on authentication and not an isolated one.

Organisation: Section 4.2.1 recalls the Fiat-Shamir authentication scheme and present a distributed algorithm for topology-aware networks. We describe our core idea, a distributed Fiat-Shamir protocol for IoT authentication, in Section 4.2.2. We analyse the security of the proposed protocol in Section 4.2.3. Section 4.2.4 provides several improvements and explores trade-offs between security, transmission and storage.

4.2.1 Preliminaries

4.2.1.1 Fiat-Shamir Authentication

The Fiat-Shamir authentication protocol [FS87] enables a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ possesses a secret key without ever revealing the secret key [GMR85, FFS88].

The algorithm first runs a one-time setup, whereby a trusted authority publishes an RSA modulus $n = pq$ but keeps the factors p and q private. The prover $\mathcal{P}$ selects a secret $s < n$ such that $\gcd(n, s) = 1$, computes $v = s^2 \mod n$ and publishes v as its public key.

When a verifier $\mathcal{V}$ wishes to identify $\mathcal{P}$, he uses the protocol of Figure 4.7. $\mathcal{V}$ may run this protocol several times until $\mathcal{V}$ is convinced that $\mathcal{P}$ indeed knows the square root s of v modulo n.

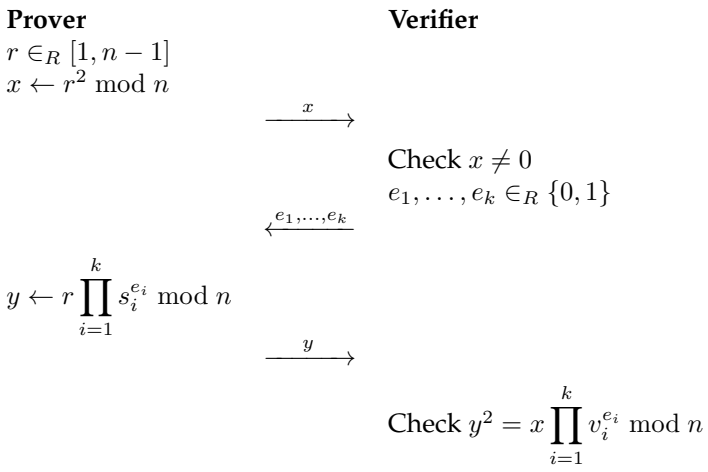


Figure 4.7: Fiat-Shamir authentication protocol.

Figure 4.7 describes the original Fiat-Shamir authentication protocol [FS87], which is *honest verifier* zero-knowledge [9], and whose security is proven assuming the hardness of computing arbitrary square roots modulo a composite n, which is equivalent to factoring n.

As pointed out by [FS87], instead of sending x, $\mathcal{P}$ can hash it and send the first bits of $H(x)$ to $\mathcal{V}$, for instance the first 128 bits. With that variant, the last step of the protocol is replaced by the computation of $H(y^2 \prod_{i=1}^{k} v_i^{-a_i} \bmod n)$, truncated to the first 128 bits, and compared to the value sent by $\mathcal{P}$. Using this "short commitment" version reduces somewhat the number of communicated bits. However, it comes at the expense of a reduced security level. A refined analysis of this technique in given in [GS94].

4.2.1.2 Topology-Aware Distributed Spanning Trees

Due to the unreliable nature of sensors, their small size and wireless communication system, the overall network topology is subject to change. Since sensors send data through the network, a sudden disruption of the usual route may result in the whole network shutting down.

4.2.1.2.1 Topology-Aware Networks. A *topology-aware* network detects changes in the connectivity of neighbours, so that each node has an accurate description of its position within the network. This information is used to determine a good route for sending sensor data to the base station. This could be implemented in many ways, for instance by sending discovery messages (to detect additions) and detecting unacknowledged packets (for deletions). Note that the precise implementation strategy does not impact the algorithm.

Given any graph $G = (V, E)$ with a distinguished vertex B (the base station), the optimal route for any vertex v is the shortest path from v to B on the minimum degree spanning tree $S = (V, E')$ of G. Unfortunately, the problem of finding such a spanning tree is **NP**-hard [SL07], even though there exist optimal approximation algorithms [SL07, LVP08]. Any spanning tree would work for the proposed algorithm, however the performance of the algorithm gets better as the spanning tree degree gets smaller.

4.2.1.2.2 Mooij-Goga-Wesselink's Algorithm. The network's topology is described by a spanning tree W constructed in a distributed fashion by the Mooij-Goga-Wesselink algorithm [MGW03]. We assume that nodes can locally detect whether a neighbour has appeared or disappeared, *i.e.* graph edge deletion and additions.

W is constructed by aggregating smaller subtrees together. Each node in W is attributed a "parent" node, which already belongs to a subtree. The complete tree structure of W is characterized by the parenthood

9. This can be fixed by requiring $\mathcal{V}$ to commit to the a_i before $\mathcal{P}$ has sent anything, but this modification will not be necessary for our purpose.

relationship, which the Mooij-Goga-Wesselink algorithm computes. Finally, by topological reordering, the base station $\mathcal{T}$ can be put as the root of W.

Each node in W has three local variables $\{\mathsf{parent}, \mathsf{root}, \mathsf{dist}\}$ that are initially set to a null value $\perp$. Nodes construct distributively a spanning tree by exchanging "M-messages" containing a root information, distance information and a type. The algorithm has two parts:

– *Basic*: maintains a spanning tree as long as no edge is removed (it is a variant of the union-find algorithm [CSRL01]). When a new neighbour w is detected, a discovery M-message $(\mathsf{root}, \mathsf{dist})$ is sent to it. If no topology change is detected for w, and an M-message is received from it, it is processed by Algorithm 6. Note that a node only becomes active upon an event such as the arriving of an M-message or a topology change.
– *Removal*: intervenes after the edge deletion so that the basic algorithm can be run again and give correct results.

Algorithm 6: Mooij-Goga-Wesselink algorithm, basic part.

Input: An M-message (r, d) coming from a neighbour w

```
1 (parent, root, dist) ← (⊥, ⊥, ⊥)
2 if (r, d + 1) < (root, dist) then
3 |   parent ← w
4 |   root ← r
5 |   dist ← d + 1
6 |   Send the M-message (root, dist) to all neighbours except w
7 end if
```

Algorithm 6 has converged once all topology change events have been processed. At that point we have a spanning tree [MGW03].

For our purposes, we may assume that the network was set up and that Algorithm 6 is running on it, so that at all times the nodes of the network have access to their parent node. Note that this incurs very little overhead as long as topology changes are rare.

4.2.2 Distributed Fiat-Shamir Authentication

4.2.2.1 The Approach

Given a k-node network $\mathcal{N}_1, ..., \mathcal{N}_k$, we may consider the nodes $\mathcal{N}_i$ as users and the base station as a trusted center $\mathcal{T}$. In this context, each node will be given only an [10] s_i. To achieve collective authentication, we propose the following Fiat-Shamir based algorithm:

– *Step 0:* Wait until the network topology has converged and a spanning tree W is constructed with Algorithm 6 presented in Section 4.2.1.2. When that happens, $\mathcal{T}$ sends an authentication request message (AR-message) to all the $\mathcal{N}_i$ directly connected to it. The AR-message may contain a commitment to e (*cf.* Step 2) to guarantee the protocol's zero-knowledge property even against dishonest verifiers.
– *Step 1:* Upon receiving an AR-message, each $\mathcal{N}_i$ generates a private r_i and computes $x_i \leftarrow r_i^2 \bmod n$. $\mathcal{N}_i$ then sends an A-message to all its children, if any. When they respond, $\mathcal{N}_i$ multiplies all the x_j sent by its children together, and with its own x_i, and sends the result up to its own parent. This recursive construction enables the network to compute the product of all the x_is and send the result x_c to the top of the tree in d steps (where $d = \deg W$). This is illustrated for a simple network including 4 nodes and a base station in Figure 4.8.
– *Step 2:* $\mathcal{T}$ sends a random e as an authentication challenge (AC-message) to the $\mathcal{N}_i$ directly connected to it.
– *Step 3:* Upon receiving an AC-message e, each $\mathcal{N}_i$ computes $y_i \leftarrow r_i s_i^{e_i}$. $\mathcal{N}_i$ then sends the AC-message to all its children, if any. When they respond, $\mathcal{N}_i$ multiplies the y_j values received from all its children together, and with its own y_i, and sends the result to its own parent. The network therefore computes collectively the product of all the y_i's and transmits the result y_c to $\mathcal{T}$. This is illustrated in Figure 4.9.

10. This is for clarity. It is straightforward to give each node several private keys, and adapt the algorithm accordingly.

– *Step 4:* Upon receiving y_c, $\mathcal{T}$ checks that $y_c^2 = x_c \prod v_i^{e_i}$, where $v_1, \ldots, v_k$ are the public keys corresponding to $s_1, \ldots, s_k$ respectively.

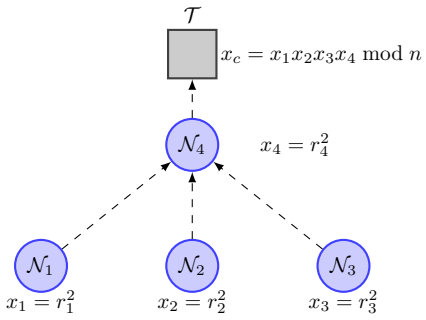


Figure 4.8: The construction of x_c.

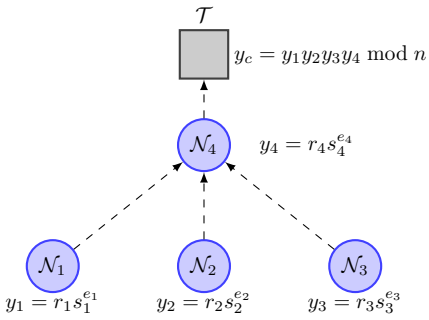


Figure 4.9: The construction of y_c.

Figure 4.10: The proposed algorithm running on a network. Each parent node aggregates the values computed by its children and adds its own information before transmitting the result upwards to the base station.

Note that the protocol may be interrupted at any step. In the version of the algorithm that we have just described, this results in a failed authentication.

4.2.2.2 Back-up Authentication

Network authentication may fail for many reasons described and analysed in detail in Section 4.2.3.3.3. As a consequence of the algorithm's distributed nature that we have just described, a single defective node suffices for authentication to fail.

This is the intended behaviour; however there are contexts in which such a brutal answer is not enough, and more information is needed. For instance, one could wish to know *which* node is responsible for the authentication failure.

A simple back-up strategy consists in performing *usual* Fiat-Shamir authentication with all the nodes that still respond, to try and identify where the problem lies. Note that, as long as the network is healthy, using our distributed algorithm instead is more efficient and consumes less bandwidth and less energy.

Since all nodes already embark the hardware and software required for Fiat-Shamir computations, and can use the same keys, there is no real additional burden in implementing this solution.

4.2.3 Security Proofs

In this section we wish to discuss the security properties relevant to our construction. The first and foremost fact is that algorithm given in Figure 4.9 is *correct*: a legitimate network will always succeed in proving its authenticity, provided that packets are correctly transmitted to the base station $\mathcal{T}$ (possibly hopping from node to node) and that nodes perform correct computations.

The interesting part, therefore, is to understand what happens when such hypotheses do *not* hold.

4.2.3.1 Soundness

Lemma 4.4 (Soundness) *If the authentication protocol of Section 4.2.2.1 succeeds then with overwhelming probability the network nodes are genuine.*

Proof: Assume that an adversary $\mathcal{A}$ simulates the whole network, but does not know the s_i, and cannot compute in polynomial time the square roots of the public keys v_i. Then, as for the original Fiat-Shamir protocol [FS87], the base station will accept $\mathcal{A}$'s identification with probability bounded by $1/2^k$ where k is the number of nodes.

□

4.2.3.2 Zero-knowledge

Lemma 4.5 (Zero-knowledge) *The distributed authentication protocol of Section 4.2.2.1 achieves statistical zero-knowledge.*

Proof: Let $\mathcal{P}$ be a prover and $\mathcal{A}$ be a (possibly cheating) verifier, who can use any adaptive strategy and bias the choice of the challenges to try and obtain information about the secret keys.

Consider the following simulator $\mathcal{S}$:

Step 1. Choose $\overline{e} \in_R \{0,1\}^k$ and $\overline{y} \in_R [0, n-1]$ using any random tape ω'

Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$ and output $(\overline{x}, \overline{e}, \overline{y})$.

The simulator $\mathcal{S}$ runs in polynomial time and outputs triples that are indistinguishable from the output of a prover that knows the corresponding private key.

If we assume the protocol is run N times, and that $\mathcal{A}$ has learnt information which we denote η, then $\mathcal{A}$ chooses adaptively a challenge using all information available to it $e(x, \eta, \omega)$ (where ω is a random tape). The proof still holds if we modify $\mathcal{S}$ in the following way:

Step 1. Choose $\overline{e} \in_R \{0,1\}^k$ and $\overline{y} \in_R [0, n-1]$ using any random tape ω'

Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$

Step 3. If $e(\overline{x}, \eta, \omega) = \overline{e}$ then go to Step 1 ; else output $(\overline{x}, \overline{e}, \overline{y})$.

Note that the protocol is also "locally" ZK, in the sense that an adversary simulating ℓ out of k nodes of the network still has to face the original Fiat-Shamir protocol.

□

4.2.3.3 Security Analysis

4.2.3.3.1 Choice of Parameters. Let λ be a security parameter. To ensure this security level the following constraints should be enforced on parameters:

- The identification protocol should be run $t \geq \lceil \lambda / k \rceil$ times (according to Lemma 4.4), which is reasonably close to one as soon as the network is large enough;
- The modulus n should take more than $2^{\lambda t}$ operations to factor;
- Private and public keys are of size comparable to n.

4.2.3.3.2 Complexity. The number of operations required to authenticate the network depends on the exact topology at hand, but can safely be bounded above:

- Number of modular squarings: $2kt$
- Number of modular multiplications $\leq 3kt$

In average, each $\mathcal{N}_i$ performs only a constant (a small) number of operations. Finally, only $O(d)$ messages are sent, where d is the degree of the minimum spanning tree of the network. Pathological cases aside, $d = O(\log k)$, so that only a logarithmic number of messages are sent during authentication.

All in all, for $\lambda = 256$, $k = 1024$ nodes and $t = 1$, we have $n \geq 2^{1024}$, and up to 5 modular operations per node.

4.2.3.3.3 Root Causes of Authentication Failure. Authentication may fail for several reasons. This may be caused by network disruption, so that no response is received from the network – at which point not much can be done.

However, more interestingly, $\mathcal{T}$ may have received an invalid value of y_c. The possible causes are easy to spot:

1. A topology change occurred during the protocol:
   – If all the nodes are still active and responding, the topology will eventually converge and the algorithm will get back to Step 0.
   – If however, the topology change is due to nodes being added or removed, the network's integrity has been altered.
2. A message was not transmitted: this is equivalent to a change in topology.
3. A node sent a wrong result. This may stem from low battery failure or when errors appear within the algorithm the node has to perform (fault injection, malfunctioning, etc). In that case authentication is expected to fail.

4.2.3.3.4 Effect of Network Noise. Individual nodes may occasionally receive incorrect (ill-formed, or well-formed but containing wrong information) messages, be it during topology reconstruction (M-messages) or distributed authentication (A-messages). Upon receiving incorrect A or M messages, nodes may dismiss them or try and acknowledge them, which may result in a temporary failure to authenticate. An important parameter which has to be taken into account in such an authentication context is the number of children of a node (fanout). When a node with many children starts failing, all its children are disconnected from the network and cannot be contacted or authenticated anymore. While a malfunction at leaf level might be benign, the failure of a fertile node is catastrophic.

4.2.3.3.5 Man-in-the-Middle. An adversary could install itself between nodes, or between nodes and the base station, and try to intercept or modify communications. Lemma 4.5 proves that a passive adversary cannot learn anything valuable, and Lemma 4.4 shows that an active adversary cannot fool authentication.

It is still possible that the adversary *relays* information, but any attempt to intercept or send messages over the network would be detected.

4.2.4 Variants and Implementation Trade-offs

The protocol may be adapted to better fit operational constraints: in the context of IoT for instance communication is a very costly operations. We describe variants that aim at reducing the amount of information sent by individual nodes, while maintaining security.

4.2.4.1 Shorter Challenges Variant

In the protocol of Section 4.2.2, the *long* (say, 128-bit) challenge e is sent throughout the network to all individual nodes. One way to reduce the length of e without compromising security is the following:

– A *short* [11] (say, 80-bit) value e is sent to the nodes;
– Each node i computes $e_i \leftarrow H(e\|i)$, and uses e_i as a challenge;
– The base station also computes e_i the same way, and uses $\{e_1, ..., e_k\}$ to check authentication.

This variant does not impact security, assuming an ideal hash function H, and it can be used in conjunction with the other improvements described below.

11. but sufficiently long in terms of entropy

4.2.4.2 Multiple Secret Variant

Instead of keeping one secret value s_i, each node could have multiple secret values $s_{i,1}, \ldots, s_{i,\ell}$. Note that these additional secrets need not be stored: they can be derived from a secret seed.

The multiple secret variant is described here for a single node, for the sake of clarity. Upon receiving a challenge e_i (assuming for instance that e_i was generated by the above procedure), each node computes a response

$$y_i \leftarrow r_i \prod_{j=1}^{\ell} s^{e_{i,j}} \bmod n$$

This can be checked by the verifier by checking whether

$$y_i^2 \stackrel{?}{=} x_i \prod_{j=1}^{\ell} v_{i,j}^{e_{i,j}} \bmod n.$$

To achieve swarm authentication, it suffices to perform aggregation as described in the protocol of Section 4.2.2 at intermediate nodes.

Using this approach, one can adjust the memory-communication trade-off, as the security level is $\lambda = t\ell$ (single-node compromission). Therefore, if $\ell = 80$ for instance, it suffices to authenticate once to get the same security as $t = 80$ authentications with $\ell = 1$ (which is the protocol of Section 4.2.2). This drastically cuts bandwidth usage, a scarce resource for IoT devices.

Furthermore, computational effort can be reduced by using batch exponentiation techniques [MN96, BGR98] to compute y_i.

4.2.4.3 Precomputed Alphabet Variant

The security level we aim at is 80 bits. A way to further reduce computational cost is the following: each node chooses an alphabet of m words $w_0, \ldots, w_{m-1}$ (a word is a 32-bit value), and computes once and for all the table of all pairwise products $p_{i,j} = m_i m_j$. Note that each $p_{i,j}$ entry is 64 bits long.

The values s_i are generated by randomly sampling from the alphabet of ws. Put differently, s_i is built by concatenating u words (bit patterns) taken from the alphabet only.

We thus see that each s_i, which is an mu-bit integer, can take m^u possible values. For instance if $m = u = 32$ then s_i is a 1024-bit number chosen amongst $32^{32} = 2^{160}$ possible values. Thanks to the lookup table, word by word multiplications need not be performed, which provides a substantial speed-up over the naive approach.

The size of the lookup table is moderate, for the example given, all we need to store is $32 \times 31/2 + 32 = 528$ values. This can be further reduced by noting that the first lines in the table can be removed: 32 values are zeros, 31 values are the results of multiplications by 1, 30 values are left shifts by 1 of the previous line, 29 values are the sum of the previous 2 and 28 values are left shifts by 2. Hence all in all the table can be compressed into $528 - 32 - 31 - 29 - 28 = 408$ entries. Because each entry is a word, this boils-down to 1632 bytes only.

4.2.4.4 Precomputed Combination Variant

Computational cost can be also cut down if we precompute and store some products, only to assemble them online during Fiat-Shamir authentication: in this variant the values of $s_{i,1,2} \leftarrow s_{i,1}s_{i,2}$, $s_{i,2,3} \leftarrow s_{i,2}s_{i,3}$, ... , etc. are stored in a lookup table.

The use of combined values $s_{i,a,b}$ in the evaluation of y results in three possible scenarios for each:

1. $s_a s_b$ appears in y – the probability of this occurring is $1/4$ – in which case one additional multiplication must be performed;
2. $s_a s_b$ does not appear in y – the probability of this occurring is $1/4$ – in which case no action is performed;

3. s_a or s_b appears, but not both – this happens with probability $1/2$ – in which case one single multiplication is required.

Consequently the expected number of multiplications is reduced by 25%, to wit $\frac{3}{4} \times 2^{m-1}$, where m is the size of e.

The method can be extended to work with a window of size $\kappa \geq 2$, for instance with $\kappa = 3$ we would precompute:

$$\begin{aligned} s_{i,3j,3j+1} &\leftarrow s_{i,3j} \times s_{i,3j+1} \\ s_{i,3j+1,3j+2} &\leftarrow s_{i,3j+1} \times s_{i,3j+2} \\ s_{i,3j,3j+2} &\leftarrow s_{i,3j} \times s_{i,3j+2} \\ s_{i,3j,3j+1,3j+2} &\leftarrow s_{i,3j} \times s_{i,3j+1} \times s_{i,3j+2} \end{aligned}$$

Following the same analysis above, the expected number of multiplications during the challenge-response phase is $\frac{7}{8} \times \frac{2^m}{3}$. The price to pay is that larger κ values claim more precomputing and more memory.

More precisely, we have the following trade-offs, writing $\mu = 2^m \bmod \kappa$:

$$\begin{aligned} \text{Multiplications (expected)} &= 2^m \left(\frac{2^\kappa - 1}{2^\kappa} \left(\left\lfloor \frac{2^m}{\kappa} - 1 \right\rfloor \right) - \frac{2^\mu - 1}{2^\mu} \right) \\ \text{Pre-multiplications} &= \ell - 1 + \left((2^\kappa - \kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor \right) + (2^\mu - \mu - 1) \\ \text{Stored Values} &= (2^\kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor + (2^\mu - 1) \end{aligned}$$

where ℓ is the number of components of s_i.

4.3 An Authenticated Encryption Scheme: Offset Merkle-Damgård

As already explained in Section 3.2, the generic composition paradigm is neither most efficient (for instance, it requires processing the input stream at least twice) nor most robust to implementation errors [Vau02, CHVV03]. To address these concerns, the notion of AE which simultaneously achieves confidentiality and integrity was put forward [KY01, BN00, BR00] and further developed [RBBK01, Rog02, Rog04a, RS06, FFL12] as a desirable primitive to be exposed by libraries and APIs to the end developer. Providing direct access to AE rather than requiring developers to make calls to several lower-level functions is seen as a step towards improving quality of security-critical code.

The scheme which will be presented next is called Offset Merkle-Damgård (OMD) and represents a keyed compression function mode of operation for nonce-based AEAD. The syntax and security notions for nonce-based AEAD schemes were formalized by Rogaway in [Rog02, Rog04a]. As a concrete example, we instantiate the OMD mode with two specific compression functions to be keyed and used, namely, the compression functions of the standard SHA-256 and SHA-512 hash functions. OMD parametrized with these two compression functions is called OMD-sha256 and OMD-sha512, respectively. The former is intended for 32-bit implementations and is the primary recommended algorithm, while the latter could be used specifically for 64-bit machines and is the secondary algorithm.

We believe that an AE scheme whose security is proved by a modular and easy to verify security reduction, only relying on some widely-verified standard assumption(s) on its underlying primitive(s), can get more confidence on its security compared to a scheme that demands strong and idealistic properties from its underlying primitive(s) or is not supported by a formal security proof. Provable security helps cryptanalysis efforts to be focused on analysing the simpler underlying primitives rather than the whole scheme.

Setting provable security in the standard model as one of our main design aims, OMD is constructed as a scheme with its security goals achieved provably, based on the sole assumption that its underlying keyed compression function is a PRF, an assumption which is among the most well-known and widely-used assumption. For example, the security of the widely-employed standard HMAC algorithm is also based on this assumption [Bel06, GPR14]. From a theoretical point of view, this is an advantage for OMD compared to the recently proposed permutation-based AE schemes in the literature whose security proofs rely on the *ideal* permutation assumption.

Unlike the mainstream AE schemes which are blockcipher-based or permutation-based schemes, OMD is designed to be a compression function based scheme. The cryptographic community has spent more than two decades on public research and standardization activities on hash functions resulting to development of a rich source of secure and efficient compression functions. Intel's announcement in July 2013 [Int13] about Intel SHA Extensions, supporting performance acceleration of the SHA family of functions (more precisely, SHA-1 and SHA-256), further encourages the decision to design a compression function based scheme. The SHA family of algorithms is heavily used in many of the most common cryptographic applications. For example, every secure web session initiation includes SHA-1, and the latest protocols involve SHA-256 as well.

The primary recommended scheme, OMD-sha256, uses the compression function of SHA-256 [FIP12]. Comparing its features with the widely known and adopted ones of the AES-GCM mode the scheme offers the following advantages:

High Quantitative Security Level. The proven security of OMD-sha256 falls off, as usual for birthday-type security bounds, in $\frac{\sigma^2}{2^{256}}$ where σ is the total number of calls to the compression function; while, for the same key size and tag size, the proven security of AES-GCM [IOM12] falls off in about $\frac{\sigma'^2}{2^{128}}$ where σ' is the total number of calls to AES. That is, with the same key length and tag length, OMD-sha256 offers higher security level than that of AES-GCM.

Flexible Key Size. AES-GCM only supports three different key lengths, namely 128, 192 and 256 bits. OMD-sha256 can support any key length between 80 bits and 256 bits.

Simple Operations. OMD-sha256 only needs the compression function of SHA-256 plus the simple operations of bitwise XOR and bitwise AND of two binary strings and (left and right) shifting a binary string. In comparison, AES-GCM in addition to calling AES requires multiplication of two arbitrary elements in GF(2^{128}). The field multiplication operation demand extra resources and is a complicated operation in contrast with the basic operations used in OMD-sha256. This is important, in particular, if one does not have access to Intel CPUs supporting the `PCLMULQDQ` instruction for implementing AES-GCM, *e.g.* on low-end devices.

Resistance Against Software-Level Timing Attacks. Most AES software implementations risk leaking their keys through cache timing [Ber05] unless they are implemented on machines with Intel CPUs supporting the constant-time AES-NI and `PCLMULQDQ` instructions. In comparison, we note that the only operations in OMD-sha256 are: bitwise XOR, AND and OR of two binary strings (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), fixed-distance (left and right) shift of a binary string (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), and 32-bit addition (of words in the compression function of SHA-256). These operations have the virtue of taking constant time on typical CPUs in which case the implementations can avoid timing attacks [12].

Organisation. The notations, definitions and concepts considered as preliminaries are presented in Section 4.3.1. Section 4.3.4 provides the specification of OMD as a mode of operation, and then the description of our two recommended instantiations: our primary recommended cipher, OMD-sha256, uses OMD with the compression function of the standard SHA-256 hash function; our secondary recommendation, OMD-sha512, uses OMD with the compression function of the standard SHA-512 hash function. We also provide the description of the compression functions of SHA-256 and SHA-512 for completeness. The security goals for OMD as an AEAD scheme are formally defined in Section 4.3.3. In Section 4.3.7, we provide the security analysis of OMD. In Section 4.3.10 we detail several interesting features of OMD. Section 4.3.2 provides an explanation of the main rationales behind the OMD design.

4.3.1 Preliminaries

Specific Notations. Let $X[i \cdots j] = X_i \cdots X_j$ denote a substring of X, for $0 \leq j \leq i \leq (m-1)$. Let $1^n 0^m$ denote concatenation of n ones by m zeros. For a non-negative integer i let $\langle i \rangle_m$ denote binary representation of i by an m-bit string.

For a binary string $X = X_{m-1} \cdots X_0$, let $X \ll n$ denote shifting X to the left by n bits (n leftmost bits are dropped and n zero bits are entered from the right); that is, $X \ll n = X_{m-n-1} \cdots X_0 0^n$. Analogously, we let $X \gg n$ denote shifting X to the right by n bits. $\neg X$ means bitwise complement of X. For two binary strings X and Y, let $X \wedge Y$ and $X \vee Y$ denote, respectively, bitwise AND and bitwise OR of the strings.

For two binary strings $X = X_{m-1} \cdots X_0$ and $Y = Y_{n-1} \cdots Y_0$, the notation $X \oplus Y$ denotes bitwise XOR of $X_{m-1} \cdots X_{m-1-\ell}$ and $Y_{n-1} \cdots Y_{n-1-\ell}$ where $\ell = \min\{m-1, n-1\}$. That is, $X \oplus Y$ is a binary string whose length is equal to the length of the shorter operand and is obtained by XORing the shorter operand with an equal length leftmost substring of the longer operand consisting of its leftmost bits. Clearly, if X and Y have the same length then $X \oplus Y$ simply means their usual bitwise XOR.

The special symbol $\perp$ means that the value of a variable is undefined; we also overload this symbol and use it to signify an error. Let $|Z|$ denote the number of elements of Z if Z is a set, and the length of Z in bits if Z is a binary string. The empty string is denoted by ε and we let $|\varepsilon| = 0$. For $X \in \{0,1\}^*$ let $X[1]||X[2]\cdots||X[m] \xleftarrow{b} X$ denote partitioning X into blocks $X[i]$ such that $|X[i]| = b$ for $1 \leq i \leq m-1$ and $|X[m]| \leq b$; let $m = |X|_b$ denote length of X in b-bit blocks.

Syntax of Keyed and Keyless Compression Functions. We denote a keyed compression function by $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$, where m and n are two positive integers, and the keyspace $\mathcal{K}$ is a non-empty set of strings. We write $F_K(H, M) = F(K; H, M)$ for every $K \in \mathcal{K}$, $H \in \{0,1\}^n$ and

12. Note that this combination of operations makes it difficult to achieve protection against power and em attacks (e.g. with masking).

$M \in \{0,1\}^m$. We can alternatively think of F_K as a single argument function whose domain is $\{0,1\}^{n+m}$ and write $F_K(H||M) = F_K(H,M)$. If $|\mathcal{K}| = 1$ we assume that $\mathcal{K} = \{\varepsilon\}$, *i.e.* it only consists of the empty string, and in this case we call F a keyless compression function. Time_F denotes the time complexity of computing $F_K(X)$ for any $K \in \mathcal{K}$ and $X \in \{0,1\}^{n+m}$, plus the time complexity for sampling from $\mathcal{K}$.

Given a keyless compression function $F' : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ (*e.g.* sha-256: $\{0,1\}^{256} \times \{0,1\}^{512} \to \{0,1\}^{256}$) we convert it to a keyed compression function F by borrowing k bits of its b-bit input block; *i.e.* we define $F_K(H,M) = F'(H,K||M)$.

Let $(\text{GF}(2^n), \oplus, .)$ denote the Galois Field with 2^n points. Any $\alpha \in \text{GF}(2^n)$ can be represented in any of the following equivalent ways:

1. As an integer between 0 and 2^n;
2. As a binary string $\alpha_{n-1} \cdots \alpha_0 \in \{0,1\}^n$;
3. As a formal polynomial $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ with binary coefficients.

For example, in $\text{GF}(2^{256})$: the string $0^{254}10$, the number 2 and the polynomial X are different representations of the same field element; the string $0^{254}11$, the number 3 and the polynomial $X+1$ represent the same field element, and so forth.

The addition "$\oplus$" and multiplication "." of two elements in $\text{GF}(2^n)$ are defined as follows. The addition of two elements $\alpha, \beta \in \text{GF}(2^n)$ simply means the element obtained by bitwise XORing their representations as binary strings. For example, $2 \oplus 1 = 0^{n-2}10 \oplus 0^{n-2}01 = 0^{n-2}11 = 3$, $2 \oplus 3 = 1$, $1 \oplus 1 = 0$, and so forth. (Note that the addition operation in $\text{GF}(2^n)$ is *different* from the addition of integers module 2^n.) To multiply two elements, first choose and fix an irreducible polynomial $P_n(X)$ of degree n over $\text{GF}(2)$; for example, choose the lexicographically first polynomial among the irreducible polynomials of degree n over $\text{GF}(2)$ with a minimum number of non-zero coefficients. For example, for $n = 256$ we use $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, for $n = 512$ we use $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$.

To multiply two elements α and β in $\text{GF}(2^n)$ denoted by $\alpha \cdot \beta$ consider them as polynomials $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ and $\beta(X) = \beta_{n-1}X^{n-1} + \cdots + \beta_1 X + \beta_0$, form their product in $\text{GF}(2)$ to get $\gamma(X)$ and take the remainder of dividing $\gamma(X)$ by the irreducible polynomial $P_n(X)$.

It is easy to multiply an arbitrary field element α by the element 2 (*i.e.* X). We describe this for $\text{GF}(2^{256})$ and $\text{GF}(2^{512})$. Let $\alpha(X) = \alpha_{n-1}X^{n-1} + \cdots + \alpha_1 X + \alpha_0$ then multiplying by X we get $\alpha_n X^n + \alpha_{n-1}X^{n-1} \cdots + \alpha_1 X + \alpha_0 X$; so if $\text{MSB}(\alpha) = 0$ then $2.\alpha = X.\alpha = \alpha \ll 1$. If $\text{MSB}(\alpha) = 1$ then we need to reduce the result by module $P_n(X)$, *i.e.* we have to add X^n to $\alpha \ll 1$. For $n = 256$ using $P_{256}(X) = X^{256} + X^{10} + X^5 + X^2 + 1$, we have $X^{256} = X^{10} + X^5 + X^2 + 1 = 0^{245}10000100101$, so adding X^{256} means XORing with $0^{245}10000100101$. For $n = 512$ using $P_{512}(X) = X^{512} + X^8 + X^5 + X^2 + 1$, we have $X^{512} = X^8 + X^5 + X^2 + 1 = 0^{503}100100101$, so adding X^{512} means XORing with $0^{503}100100101$. In summary, for $\text{GF}(2^{256})$

$$2.\alpha = \begin{cases} \alpha \ll 1 & \text{if MSB}(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{245}10000100101 & \text{if MSB}(\alpha) = 1 \end{cases}$$

and for $\text{GF}(2^{512})$

$$2.\alpha = \begin{cases} \alpha \ll 1 & \text{if MSB}(\alpha) = 0 \\ (\alpha \ll 1) \oplus 0^{503}100100101 & \text{if MSB}(\alpha) = 1 \end{cases}$$

4.3.2 Design Rationale

Provable security. We aimed to have a scheme with a sound security guarantee in the style of reduction-based provable security relying only on a single well-established standard assumption on the underlying primitive, namely the PRF assumption on the keyed compression function. The security goals of privacy and authenticity for OMD are achieved provably; that is, any attack against these security goals will imply an attack against the classical PRF property of the underlying compression function. We note that any good keyed compression function (either a dedicated-key one or keyed via some part of its input) must provide the classical PRF property when its key is secret as otherwise it will be considered useless for almost any secret key application, *e.g.* for being used as the compression function of a hash function in the standard HMAC algorithm. That is, the base PRF assumption on the compression function upon

which the security of OMD relies is highly assured for compression functions of the practical, standard hash functions, thanks to the vast amount of cryptanalytic work on these functions.

Simple structure. Simplicity is important in any cryptographic algorithm: the easier an algorithm is to understand, the easier it is to analyse and to get confidence on its security, and also less prone it is to implementation errors. Therefore, simplicity was one of our core design goals. The high level structure of OMD is quite simple and resembles the well-known structures for hash functions and MACs, namely, the part that is processing the message resembles the Merkle-Damgård iteration where at each iteration random bits are derived from the chaining values to be used for encryption and a key-dependent offset value is XORed to the chaining values. The part for processing the associated data is inspired by the XMACC scheme (counter-based XOR MAC scheme) [BGR95] and is a simple adaptation of the similar hashing process in the OCB3 algorithm [KR11]. We note that when the message is empty then OMD acts almost the same as XMACC on the associated data.

Attacks. Any attack against security of OMD means an attack against the specific compression function that is used for instantiating OMD. For example, attacking OMD-sha256 will imply attacking the compression function of SHA-256 in the PRF sense.

4.3.3 Security Definitions and Goals

OMD is a nonce-based authenticated encryption scheme with associated data (AEAD), thus it aims at achieving the security notions for AEAD schemes as presented in Section 3.2.1.

Note. OMD requires the nonce-respecting condition: it does not provide security if the nonce is repeated. We have not analyzed at which extent the security is lost so far. Our design rationale stemmed from the security notions formalized in [Rog02, Rog04b] and the liaison with the OCB mode of operation [RBBK01].

4.3.3.1 Quantitative Security Level of OMD-sha256/512

Let $n = 256$ (the hash size in the case of sha-256) or $n = 512$ (the hash size in the case of sha-512). Based on the concrete security bounds given in Section 4.3.7, we can compute the quantitative security (privacy and authenticity) levels of OMD-sha256/512 for any set of fixed values for the adversarial resource parameters. For this purpose, we make the assumption that the function $F_K(H, M) = \text{sha-256/512}(H, K||0^{256/512-k}||M)$ is a PRF providing a k-bit security; as (to the best of our knowledge) there is no known attack with complexity less than 2^k against it. We note that having only a single (input, output) pair for F_K one can mount an *off-line* exhaustive search attack with time complexity 2^k.

For the privacy property of OMD-sha256/512 (*i.e.* "confidentiality for the plaintext") the security bound falls off in $\frac{3\sigma_e^2}{2^{256/512}}$. That is, if the adversary has *on-line* data complexity about $\sigma_e = 2^{127/255}$, where σ_e denotes the total number of blocks in all inputs for encryption and decryption as defined in Section 4.3.3. We note that, giving a single measure for the bit security level of OMD-sha256 is a bit tricky as the terms determining the security bound and the resources are different in nature (e.g. we have both off-line complexity and on-line complexity); nevertheless, one can roughly consider $\min\{k, 127/255\}$ as the bit security.

For the authenticity property of OMD-sha256/512 (*i.e.* "integrity for the public message number, the associated data and the plaintext") the security bound falls off in

$$\frac{3\sigma^2}{2^{256/512}} + \frac{q_v \ell_{\max}}{2^{256/512}} + \frac{q_v}{2^\tau}.$$

That is, if the adversary has *on-line* data complexity about $\sigma_e = 2^{127/255}$, or $q_v \ell_{\max} = 2^{256/512}$, or $q_v = 2^\tau$ (we refer to Section 4.3.3 for definitions of the resource parameters). As a single measure for the bit security of OMD-sha256/512 for the authenticity goal, one can roughly consider $\min\{k, 127/255, \tau\}$.

Note. We note that a single measure for the "bit security level" should be interpreted carefully regarding the different on-line/off-line nature of the resources used for complexity measures. For example, just based on our bit security levels for OMD-sha256/512 one may think that a key length (k) larger than 127/255 bits or larger than the tag length (τ) is not useful, but this is not true because, for example, while the role of τ is to prevent on-line attacks, a large k can help prevent (mainly) off-line key recovery attacks (that may only use one on-line query).

4.3.4 Specification of OMD

4.3.4.1 The OMD Mode of Operation

OMD is a compression function mode of operation for nonce-based AEAD. To use OMD one must specify a keyed compression function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ and a tag length $\tau \leq n$, where the key space $\mathcal{K} = \{0,1\}^k$ and $m \leq n$ the case $m = n$ is the optimal choice from efficiency viewpoint. At first glance, requiring $m \leq n$ may look a bit odd as usually a compression function has a larger input block length than its output (hash) length, so we first explain this restriction based on the following two observations:

- The following description of OMD will clarify that at each call to the compression function only n random bits (namely, the output bits of the compression function) are available for encrypting an m-bit message block, hence we must have $m \leq n$. The optimal case is when $m = n$, so no random bits are wasted. We notice that this limitation applies to any compression function based AE, therefore a compression function based AE scheme (like OMD) will be usually less efficient than a blockcipher based AE (like OCB) "unless" one uses a dedicated compression function which is more efficient than the blockcipher.
- In practice, the compression function of standard hash functions (e.g. SHA-1 or the SHA-2 family) are keyless, *i.e.* do not have a dedicated key input, therefore one will need to use k bits of their b-bit message block to get a keyed function. So, there will be no efficiency waste in each call to the compression function if $m = n$ and $b = n + k$. For example, when the key length is 256 bits and the compression function of SHA-256 is used.

We let OMD-F denote the OCB mode of operation using a keyed compression function $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ with $m \leq n$ and an unspecified tag length. We let OMD[F, τ] denote the OMD mode of operation using keyed compression function F_K and tag length τ. The encryption algorithm Enc-OMD[F, τ] inputs four arguments (secret key $K \in \{0,1\}$, nonce $N \in \{0,1\}^{|N|}$, associated data $A \in \{0,1\}^*$, message $M \in \{0,1\}^*$) and outputs $\mathbb{C} = C||\text{Tag} \in \{0,1\}^{|M|+\tau}$. The decryption algorithm Dec-OMD[F, τ] inputs four arguments (secret key $K \in \{0,1\}$, nonce $N \in \{0,1\}^{|N|}$, associated data $A \in \{0,1\}^*$, ciphertext $C||\text{Tag} \in \{0,1\}^*$) and either outputs the whole $M \in \{0,1\}^{|C|-\tau}$ at once or an error message ($\perp$). Note that we have either $C = C_1 \cdots C_\ell$ or $C = C_1 \cdots C_{\ell-1} C_*$ depending on whether the message length in bits is a multiple of the block length m or not, respectively.

Figure 4.11 depicts the construction of Enc-OMD[F, τ]. The construction of Dec-OMD[F, τ] is straightforward and almost the same as Enc-OMD[F, τ] except a tag comparison (verification) at the end of decryption. In Algorithms 9 and 10 we describe the encryption and decryption algorithms of OMD-F. We remind the reader that for two binary strings $X = X_{m-1} \cdots X_0$ and $Y = Y_{n-1} \cdots Y_0$, the notation $X \oplus Y$ denotes bitwise XOR of $X_{m-1} \cdots X_{m-1-\ell}$ and $Y_{n-1} \cdots Y_{n-1-\ell}$ where $\ell = \min\{m-1, n-1\}$.

Computing the Masking Values. As seen from the description of OMD in Figure 4.11, before each call to the underlying keyed compression function we XOR a masking value denoted as $\Delta_{N,i,j}$ (the top and middle parts of Figure 4.11) and $\bar{\Delta}_{i,j}$ (the bottom part of Figure 4.11). In the following, we describe how these masks are generated. We note that there are both security and efficient related criteria to be satisfied by the method to compute the masking values. We only explain the efficiency criterion for computing the masks here; the security related properties will be made clear in Section 4.3.7. By an efficient masking scheme, we mean a scheme in which the mask value needed for processing a block can be efficiently computed from the mask value used for processing the previous block.

There are different ways to compute the masking values to satisfy both the security and efficiency criteria. For example, we refer to [Rog04a, CS08, KR11]. We use the method proposed in [KR11].

In the following, all multiplications are performed in GF(2^n), $\mathtt{ntz}(i)$ denotes the number of trailing zeros (*i.e.* the number of rightmost bits that are zero) in the binary representation of a positive integer i.

Initialization.

$$\begin{cases} \Delta_{N,0,0} & \leftarrow & F_K(N||10^{n-1-|N|},0^m) \\ \bar{\Delta}_{0,0} & \leftarrow & 0^n \\ L_* & \leftarrow & F_K(0^n,0^m) \\ L[0] & \leftarrow & 4L_* \\ L[i] & \leftarrow & 2L[i-1] \text{ for } i \geq 1 \end{cases}$$

We note that the values $L[i]$ can be preprocessed and stored (for a fast implementation) in a table for $0 \leq i \leq \lceil \log_2(\ell_{\max}) \rceil$, where $\ell_{\max}$ is the the bound on the maximum number of m-bit blocks in any message that can be encrypted or decrypted. Alternatively (if there is a memory restriction), they can be computed on-the-fly for $i \geq 1$.

Masking Sequence for Processing the Message. For $i \geq 1$:

$$\begin{cases} \Delta_{N,i,0} & \leftarrow & \Delta_{N,i-1,0} \oplus L[\mathtt{ntz}(i)] \\ \Delta_{N,i,1} & \leftarrow & \Delta_{N,i,0} \oplus 2L_* \\ \Delta_{N,i,2} & \leftarrow & \Delta_{N,i,0} \oplus 3L_* \end{cases}$$

Masking Sequence for Processing the Associated Data.

$$\begin{cases} \bar{\Delta}_{i,0} & \leftarrow & \bar{\Delta}_{i-1,0} \oplus L[\mathtt{ntz}(i)] \text{ for } i \geq 1 \\ \bar{\Delta}_{i,1} & \leftarrow & \bar{\Delta}_{i,0} \oplus L_* \text{ for } i \geq 0 \end{cases}$$

4.3.5 OMD-SHA256: Primary Recommendation for Instantiating OMD

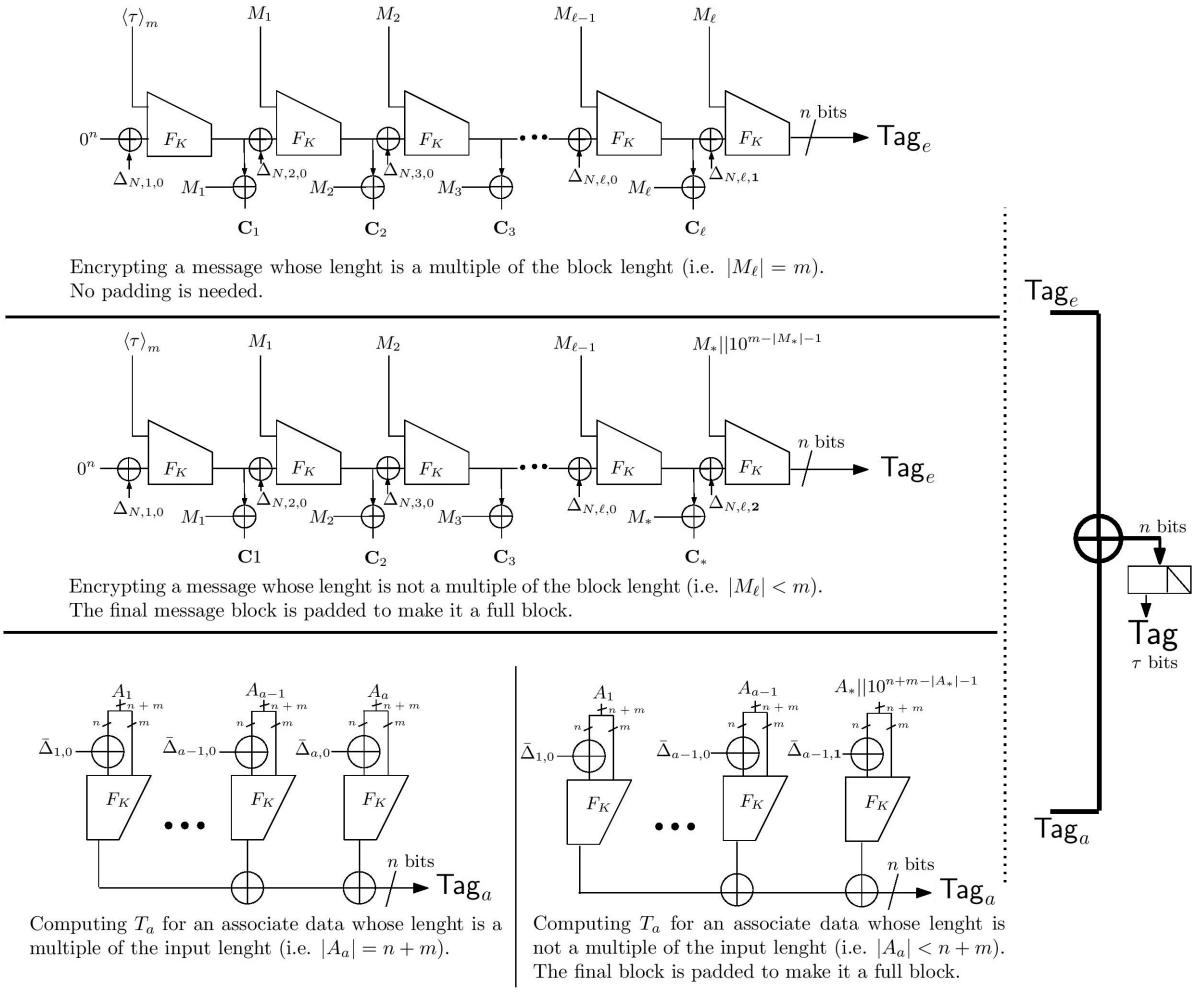
Our primary recommendation to instantiate OMD is called OMD-sha256 and uses the underlying compression function of SHA-256 [FIP12]. The compression function of SHA-256 is a map sha-256 : $\{0,1\}^{256} \times \{0,1\}^{512} \rightarrow \{0,1\}^{256}$. On input a 256-bit chaining block X and a 512-bit message block Y, it outputs a 256-bit digest Z, *i.e.* let $Z =$ sha-256(X,Y). The description of sha-256 is provided in Section A.1.

To use OMD with sha-256, we use the first 256-bit argument X for chaining values as usual. In our notation (see Figure 4.11) this means that $n = 256$. We use the 512-bit argument Y (the message block in sha-256) to input both a 256-bit message block and the key K which can be of any length $k \leq 256$ bits. If $k < 256$ then let the key be $K||0^{256-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{256} \times \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ needed in OMD as $F_K(H,M) =$ sha-256$(H, K||0^{256-k}||M)$.

The parameters of OMD-sha256 are as follows:

- The message block length in bits is $m = 256$; *i.e.* $|M_i| = 256$. *If needed*, we pad the final block of the message with 10^* (*i.e.*, a single 1 followed by the minimal number of 0's needed) to make its length exactly 256 bits.
- The key length in bits can be $80 \leq k \leq 256$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{256-k} to make its length exactly 256 bits.
- The nonce (public message number) length in bits can be $96 \leq |N| \leq 255$. We *always* pad the nonce with $10^{255-|N|}$ to make its length exactly 256 bits.
- The secret message number [13] length in bits is 0; that is, our scheme does not support secret message numbers.
- The associated data block length in bits is $2n = 512$; *i.e.* $|A_i| = 512$. *If needed*, we pad the final block of the associated data with 10^* (*i.e.*, a single 1 followed by the minimal number of 0's needed) to make its length exactly 512 bits.

13. The secret message number is a secret nonce; it must be unique for every encryption. This parameter was proposed by the CAESAR committee in order to add flexibility and security, mostly for multiple-message network protocols.



Encrypting a message whose lenght is a multiple of the block lenght (i.e. $|M_\ell| = m$). No padding is needed.

Encrypting a message whose lenght is not a multiple of the block lenght (i.e. $|M_\ell| < m$). The final message block is padded to make it a full block.

Computing T_a for an associate data whose lenght is a multiple of the input lenght (i.e. $|A_a| = n + m$).

Computing T_a for an associate data whose lenght is not a multiple of the input lenght (i.e. $|A_a| < n + m$). The final block is padded to make it a full block.

Figure 4.11: The encryption process of OMD-(F, τ) using a keyed compression function $F_K : (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ with $m \leq n$. (**Top**) The encryption process when the message length is a multiple of the block length m and no padding is required. (**Middle**) The encryption process when the message length is not a multiple of the block length and the final block M_* is padded to make a full block $M_*||10^{m-|M_*|-1}$. (**Bottom, Left**) Computing the intermediate value T_a when the bit length of the associated data is a multiple of the **input** length $n + m$. (**Bottom, Right**) Computing T_a when the bit length of the associated data is not a multiple of $n + m$ and the final block is padded to make a full block $A_*||10^{n+m-|A_*|-1}$ is needed. The output ciphertext is $\mathbf{C}||\mathrm{Tag}$. For operation $\oplus$ see our convention in Section 4.3.1. Five types of masking values (corresponding to five mutually exclusive tweak sets) are used; these are denoted by $\Delta_{N,i,0}, \Delta_{N,i,1}, \Delta_{N,i,2}, \bar{\Delta}_{i,0}$ and $\bar{\Delta}_{j,1}$, for $i \geq 1$ and $j \geq 0$, where N is the nonce. Note that the masks used in computing T_a do not depend on the nonce.

– The tag length in bits can be $32 \leq \tau \leq 256$, but it must be noted that the selection of the tag length directly affects the achievable security level. We refer to Section 4.3.7 for the security bounds.

Definition of OMD[F, τ]. The function $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \rightarrow \{0,1\}^n$ is a keyed compression function with $\mathcal{K} = \{0,1\}^k$ and $m \leq n$. The tag length is $\tau \in \{0, 1, \cdots, n\}$. Algorithms $\mathcal{E}$ and $\mathcal{D}$ can be called with arguments $K \in \mathcal{K}$, $N \in \{0,1\}^{\leq n-1}$, and $A, M, \mathbb{C} \in \{0,1\}^*$. $\ell_{\max}$ is the bound on the maximum number of blocks in any input to the encryption or decryption algorithms.

Algorithm 7: INITIALIZE (K)

$L_* \leftarrow F_K(0^n, 0^m)$
$L[0] \leftarrow 4.L_*$ ▷ $2.(2.L_*)$, doubling in GF(2^n)
for $i \leftarrow 1$ to $\lceil \log_2(\ell_{\max}) \rceil$ **do**
  $L[i] = 2.L[i-1]$ ▷ doubling in GF(2^n)
end for
return

Algorithm 8: HASH$_K(A)$

$b \leftarrow n + m$
$A_1 || A_2 || \cdots || A_{\ell-1} || A_\ell \xleftarrow{b} A$, where $|A_i| = b$ for $1 \leq i \leq \ell - 1$ and $|A_\ell| \leq b$
$\mathrm{Tag}_a \leftarrow 0^n$
$\Delta \leftarrow 0^n$
for $i \leftarrow 1$ to $\ell - 1$ **do**
  $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i)]$
  $\mathsf{Left} \leftarrow A_i[b-1, \cdots, m]$
  $\mathsf{Right} \leftarrow A_i[m-1, \cdots, 0]$
  $\mathrm{Tag}_a \leftarrow \mathrm{Tag}_a \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
end for
if $|A_\ell| = b$ **then**
  $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(\ell)]$
  $\mathsf{Left} \leftarrow A_\ell[b-1, \cdots, m]$
  $\mathsf{Right} \leftarrow A_\ell[m-1, \cdots, 0]$
  $\mathrm{Tag}_a \leftarrow \mathrm{Tag}_a \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
end if
else
  $\Delta \leftarrow \Delta \oplus L_*$
  $\mathsf{Left} \leftarrow A_\ell || 10^{b-|A_\ell|-1}[b-1, \cdots, m]$
  $\mathsf{Right} \leftarrow A_\ell || 10^{b-|A_\ell|-1}[m-1, \cdots, 0]$
  $\mathrm{Tag}_a \leftarrow \mathrm{Tag}_a \oplus F_K(\mathsf{Left} \oplus \Delta, \mathsf{Right})$
end if
return Tag_a

4.3.6 OMD-SHA512: Secondary Recommendation for Instantiating OMD

Our secondary recommendation to instantiate OMD is called OMD-sha512 and uses the underlying compression function of SHA-512 [FIP12]. The compression function of SHA-512 is a map sha-512 : $\{0,1\}^{512} \times \{0,1\}^{1024} \rightarrow \{0,1\}^{512}$. On input a 512-bit chaining block X and a 1024-bit message block

Algorithm 9: $\mathcal{E}_K(N, A, M)$

if $|N| > n - 1$ **then**
  return
end if
$\perp M_1||M_2||\cdots||M_{\ell-1}||M_\ell \xleftarrow{m} M$, where $|M_i| = m$ for $1 \le i \le \ell - 1$ and $|M_\ell| \le m$
$\Delta \leftarrow F_K(N||10^{n-1-|N|}, 0^m)$ ▷ initialize $\Delta_{N,0,0}$
$H \leftarrow 0^n$
$\Delta \leftarrow \Delta \oplus L[0]$ ▷ compute $\Delta_{N,1,0}$
$H \leftarrow F_K(H \oplus \Delta, \langle\tau\rangle_m)$
for $i \leftarrow 1$ to $\ell - 1$ **do**
  $C_i \leftarrow H \oplus M_i$
  $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i+1)]$
  $H \leftarrow F_K(H \oplus \Delta, M_i)$
end for
$C_\ell \leftarrow H \oplus M_\ell$ **if** $|M_\ell| = m$ **then**
  $\Delta \leftarrow \Delta \oplus 2.L_*$
  $\mathrm{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
end if
else
  if $|M_\ell| \neq 0$ **then**
    $\Delta \leftarrow \Delta \oplus 3.L_*$
    $\mathrm{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell||10^{m-|M_\ell|-1})$
  end if
end if
else
  $\mathrm{Tag}_e \leftarrow H$
end if
$\mathrm{Tag}_a \leftarrow \textsc{Hash}_K(A)$
$\mathrm{Tag} \leftarrow (\mathrm{Tag}_e \oplus \mathsf{Tag}_a)[n-1, \cdots, n-\tau]$
$\mathbb{C} \leftarrow C_1||C_2||\cdots||C_\ell||\mathsf{Tag}$ **return** $\mathbb{C}$

Y, it outputs a 512-bit digest Z, *i.e.* let $Z = \text{sha-512}(X, Y)$. The description of sha-512 is provided in Section A.1.

To use OMD with sha-512, we use the first 512-bit argument X for chaining values as usual. In our notation (see Figure 4.11) this means that $n = 512$. We use the 1024-bit argument Y (the message block in sha-512) to input both a 512-bit message block and the key K which can be of any length $k \le 512$ bits. If $k < 512$ then let the key be $K||0^{512-k}$. That is, we define the keyed compression function $F_K : \{0,1\}^{512} \times \{0,1\}^{512} \to \{0,1\}^{512}$ needed in OMD as $F_K(H, M) = \text{sha-512}(H, K||0^{512-k}||M)$.

The parameters of OMD-sha512 are set as follows:

- The message block length in bits is $m = 512$; *i.e.* $|M_i| = 512$. *If needed*, we pad the final block of the message with 10^* (*i.e.*, a single 1 followed by the minimal number of 0's needed) to make its length exactly 512 bits.
- The key length in bits can be $80 \le k \le 512$; but $k < 128$ is not recommended. *If needed*, we pad the key K with 0^{512-k} to make its length exactly 512 bits.
- The nonce (public message number) length in bits can be $96 \le |N| \le 511$. We *always* pad the nonce with $10^{511-|N|}$ to make its length exactly 512 bits.
- The secret message number length in bits is 0; that is, our scheme does not support secret message numbers.
- The associated data block length in bits is $2n = 1024$; *i.e.* $|A_i| = 1024$. *If needed*, we pad the final block

of the associated data with 10^* (*i.e.*, a single 1 followed by the minimal number of 0's needed) to make its length exactly 1024 bits.
– The tag length in bits can be $32 \leq \tau \leq 512$, but it must be noted that the selection of the tag length directly affects the achievable security level. We refer to Section 4.3.7 for the security bounds.

4.3.7 Security Proofs

Theorem 4.6 provides the security bounds of OMD.

Theorem 4.6 *Fix $n \geq 1$ and $\tau \in \{0,1,\cdots,n\}$. Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a PRF, where the key space $\mathcal{K} = \{0,1\}^k$ for $k \geq 1$ and $1 \leq m \leq n$. Then*

$$\begin{aligned} \mathrm{Adv}^{\mathrm{priv}}_{\mathrm{OMD}[F,\tau]}(t, q_e, \sigma_e, \ell_{\max}) &\leq \mathrm{Adv}^{\mathrm{prf}}_{F}(t', 2\sigma_e) + \frac{3\sigma_e^2}{2^n} \\ \mathrm{Adv}^{\mathrm{auth}}_{\mathrm{OMD}[F,\tau]}(t, q_e, q_v, \sigma, \ell_{\max}) &\leq \mathrm{Adv}^{\mathrm{prf}}_{F}(t', 2\sigma) + \frac{3\sigma^2}{2^n} + \frac{q_v \ell_{\max}}{2^n} + \frac{q_v}{2^\tau} \end{aligned}$$

where q_e and q_v are, respectively, the number of encryption and decryption queries, ℓ_{max} denotes the maximum number of m-bit blocks in an encryption or decryption query, $t' = t + cn\sigma$ for some constant c, and σ_e and σ are the total number of calls to the underlying compression function F in all queries asked by the CPA *and* CCA *adversaries against the privacy and authenticity of the scheme, respectively.*

The proof is obtained by combining Lemma 4.7 in Section 4.3.8 with Lemma 4.8 and Lemma 4.9 in Section 4.3.9.

Note. Referring to Section 4.3.3 for definitions of the resource parameters, it can be seen that: $\sigma_e = \lceil \sigma_M/m \rceil + \lceil \sigma_A/(n+m) \rceil + q_e + 2$; $\sigma = \lceil (\sigma_M + \sigma_{\mathbb{C}'})/m \rceil + \lceil (\sigma_A + \sigma_{A'})/(n+m) \rceil + q + 2$; and $\ell_{\max} = \lceil L_{\max}/m \rceil$.

4.3.8 Generalisation of OMD Based on Tweakable Random Functions

Figure 4.12 shows the $\mathbb{OMD}[\widetilde{R}, \tau]$ scheme which is a generalization of OMD$[F, \tau]$ using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$. The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks, namely $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$, where $\mathcal{N} = \{0,1\}^{|N|}$ is the set of nonces and $\mathbb{N}$ is the set of positive integers.

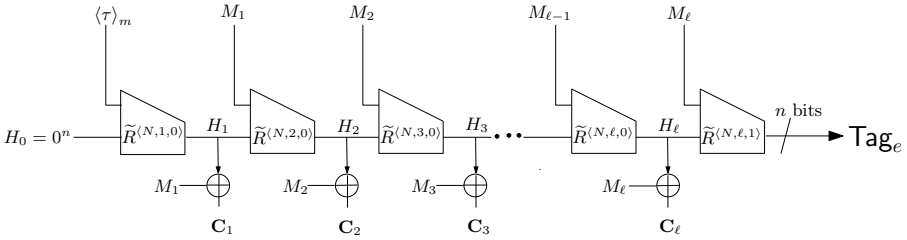
Lemma 4.7 *Let $\mathbb{OMD}[\widetilde{R}, \tau]$ be the scheme shown in Figure 4.12. Then*

$$\begin{aligned} \mathrm{Adv}^{\mathrm{priv}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e, \sigma_e, \ell_{\max}) &= 0 \\ \mathrm{Adv}^{\mathrm{auth}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e, q_v, \sigma, \ell_{\max}) &\leq \frac{q_v \ell_{\max}}{2^n} + \frac{q_v}{2^\tau}\,. \end{aligned}$$

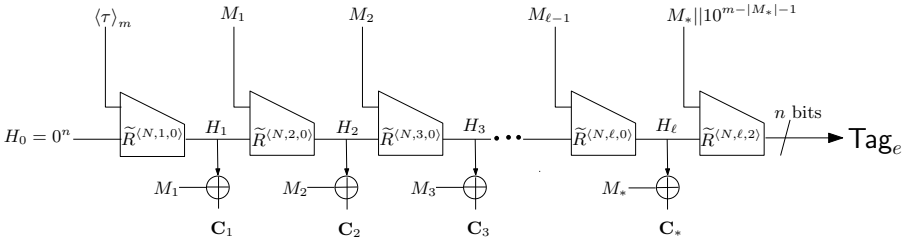
where q_e and q_v are, respectively, the number of encryption and decryption queries, $\ell_{\max}$ denotes the maximum number of m-bit blocks in an encryption or decryption query, and σ_e and σ are the total number of calls to the underlying tweakable random function $\widetilde{R}$ in all queries asked by the CPA *and* CCA *adversaries against the privacy and authenticity of the scheme, respectively.*

The proof of the privacy bound is straightforward. Let $\mathcal{A}$ be a CPA adversary that asks (encryption) queries $(N^1, A^1, M^1), \cdots, (N^{q_e}, A^{q_e}, M^{q_e})$ where all N^x values (for $1 \leq x \leq q_e$) are distinct due to the nonce-respecting assumption on the adversary $\mathcal{A}$. Referring to Figure 4.12, this means that we are applying independent random functions $\widetilde{R}^{N_x,i,j}$ each to a single point, hence the images that the adversary sees (*i.e.* $\mathbb{C}^x$ for $1 \leq x \leq q_e$) are fresh uniformly random values.

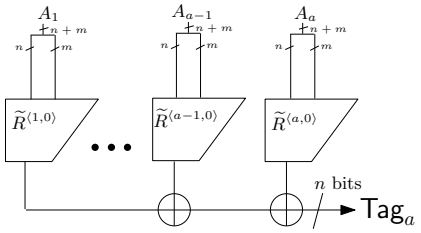
The authenticity bound can be shown by a straightforward but lengthy case analysis. First we consider the single verification case where the adversary only makes one decryption (verification) query and then we will use the generic result of Bellare *et al.* [BGM04] to get a bound against adversaries that



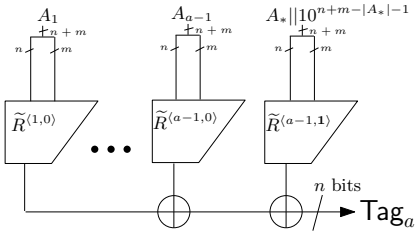
Encrypting a message whose length is a multiple of the block length. No padding is needed.



Encrypting a message whose length is not a multiple of the block length. The final message block is padded to make it a full block.



Computing Tag_a for an associate data whose length is a multiple of the input length (i.e. $|A_a| = n + m$).



Computing Tag_a for an associate data whose length is not a multiple of the input length. The final block is padded to make it a full block.

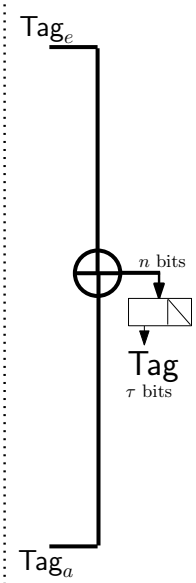


Figure 4.12: The $\mathbb{OMD}[\widetilde{R}, \tau]$ scheme using a tweakable random function $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ (*i.e.* $\widetilde{R} \xleftarrow{R} \mathrm{Func}^{\mathcal{T}}(n+m, n)$). The tweak space $\mathcal{T}$ consists of five mutually exclusive sets of tweaks, namely $\mathcal{T} = \mathcal{N} \times \mathbb{N} \times \{0\} \cup \mathcal{N} \times \mathbb{N} \times \{1\} \cup \mathcal{N} \times \mathbb{N} \times \{2\} \cup \mathbb{N} \times \{0\} \cup \mathbb{N} \times \{1\}$, where $\mathcal{N} = \{0,1\}^{|N|}$ is the set of nonces, $\mathbb{N}$ is the set of positive integers.

make multiple (say q_v) verification queries. Let $\mathcal{A}$ be a CCA adversary making encryption queries $(N^1, A^1, M^1), \cdots, (N^{q_e}, A^{q_e}, M^{q_e})$. Let $M^i = M^i_1, \cdots, M^i_{\ell_i}$ or $M^i = M^i_1, \cdots, M^i_{\ell_i - 1} M^i_*$ be the message queries and $A^i = A^i_1, \cdots, A^i_{a_i}$ or $A^i = A^i_1, \cdots, A^i_{a_i - 1} A^i_*$ be the associated data queries. Let $\mathbb{C}^i = C^i || \mathsf{Tag}^i$ be the ciphertext received for query (N^i, A^i, M^i). That is, we use superscripts to indicate query numbers and subscripts to denote the block indices in each query.

Let $(N, A, \mathbb{C})$ be the forgery attempt by the adversary, where $N \in \{0,1\}^{|N|}$ is the nonce, $A = A_1, \cdots, A_a$ or $A = A_1, \cdots, A_{a-1} A_*$ is the associate data, $\mathbb{C} = C || \mathsf{Tag}$ is the ciphertext where $C = C_1, \cdots, C_\ell$ (where $|C_i| = m$ for $1 \leq i \leq \ell$) or $C = C_1, \cdots, C_{\ell-1} C_*$ (where $|C_i| = m$ for $1 \leq i \leq \ell - 1$ and $|C_*| < m$), and $\mathsf{Tag} = (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1, \cdots, n-\tau] \in \{0,1\}^\tau$ is the tag. Let $M = M_1, \cdots, M_\ell$ or $M = M_1, \cdots, M_{\ell-1} M_*$ denote the corresponding decrypted messages, respectively. Note that no superscripts are used for the strings in the alleged forgery by the adversary. We have the following disjoint cases:

① $N \notin \{N^1, \cdots, N^{q_e}\}$. Adversary has to find a correct Tag that is the first τ bits of the value $\widetilde{R}^{\langle N,x,y \rangle}(\text{final input}) \oplus \mathsf{Tag}_a$ but has not seen any image under $\widetilde{R}^{\langle N,x,y \rangle}(.)$, hence the probability that the adversary can succeed in doing this is $2^{-\tau}$. By "final input" we mean $H_\ell || M_\ell$ or $H_\ell || M_* || 10^{m-|M_*|-1}$ when $|C| \neq 0$ in which case the final tweak used to generate Tag_e will be either $\langle N, \ell, 1 \rangle$ or $\langle N, \ell, 2 \rangle$ (depending on whether the final block is a full block or not); otherwise (*i.e.* for empty message) the "final input" will be $H_0 || \langle \tau \rangle_m$ and hence the final tweak used to generate Tag_e will be $\langle N, 1, 0 \rangle$.

② $N = N^i$, $|C| \neq |C^i|$, and one of $|C|$ and $|C^i|$ is a multiple of m but the other is not. We can ignore all queries other than the i^{th} query since the responses to such queries are random and unrelated (because of using different nonces) to the adversary's task to make the alleged forgery $N, A, \mathbb{C}$ with $N = N^i$. That is, we can assume that adversary has only made a single encryption query (N^i, A^i, M^i) and received $C^i || \mathsf{Tag}^i$. Then as in Case 1 the adversary has to find a correct Tag, *i.e.* the first τ bits of the value $\widetilde{R}^{\langle N,x,y \rangle}(\text{final input}) \oplus \mathsf{Tag}_a$, but has not seen any image under $\widetilde{R}^{\langle N,x,y \rangle}(.)$. Note that we can even give Tag_a to the adversary. More precisely, consider the case that $|C^i|$ is a multiple of m but $|C|$ is not; then adversary must guess the first τbits of the value $\widetilde{R}^{\langle N,\ell,2 \rangle}(\text{final input}) \oplus \mathsf{Tag}_a$, but has seen no image under $\widetilde{R}^{\langle N,\ell,2 \rangle}(.)$. Similarly, in the case that $|C|$ is a multiple of m but $|C^i|$ is not, the adversary must guess the first τ bits of the value $\widetilde{R}^{\langle N,x,y \rangle}(\text{final input}) \oplus \mathsf{Tag}_a$ for $(N, x, y) = (N, 1, 0)$ if $|C| = 0$ or $(N, x, y) = (N, \ell, 1)$ if $|C| \neq 0$, but the adversary has seen no image under $\widetilde{R}^{\langle N,x,y \rangle}(.)$ under either case.Therefore, the probability that the adversary can succeed in guessing Tag is $2^{-\tau}$.

③ $N = N^i$, $|C| \neq |C^i|$, and either both $|C|$ and $|C^i|$ are multiple of m or none of them is. We may ignore all queries but the i^{th} query as responses to such queries are unrelated to the adversary's task at hand. If both $|C|$ and $|C^i|$ are multiple of m then $|C| \neq |C^i|$ means that $\ell \neq \ell^i$, so from (the top of) Figure 4.12 it can be easily seen that in this case even if the adversary knows Tag_a it must still guess the first τ bits of the output of the random function $\widetilde{R}^{\langle N,\ell,1 \rangle}$ while it has seen no image of this function; the probability to succeed in guessing Tag is clearly $2^{-\tau}$. Now, let's consider the case that neither $|C|$ nor $|C^i|$ is a multiple of m then $|C| \neq |C^i|$ means that we have two cases: (1) $\ell \neq \ell^i$, and (2) $\ell = \ell^i$ but $|C_*| \neq |C^i_*|$. In the first case, it can be seen the adversary must guess the first τ bits of the random function $\widetilde{R}^{\langle N,\ell,2 \rangle}$ while has seen no image of this function; the chance to do so is clearly $2^{-\tau}$. In the second case, the adversary must guess the first τ bits of $\widetilde{R}^{\langle N,\ell,2 \rangle}((M_* \oplus C_*) || (M_* || 10^{m-|M_*|-1}))$ while it has seen (τ bits of) a single image of this function for one different domain point, namely $((M^i_* \oplus C^i_*) || (M^i_* || 10^{m-|M^i_*|-1})$; the probability to succeed in this case is again $2^{-\tau}$. (Note that $|M_*| = |C_*|$. Using 10^* padding for processing messages whose length is not a multiple of m is essential for this part.)

④ $N = N^i$, $|C| = |C^i|$, and $A \neq A^i$. We can ignore all queries except the i^{th} query because the responses to such queries are random and unrelated to the adversary's task to make the alleged forgery $N, A, \mathbb{C}$ with $N = N^i$. That is, we can assume that adversary has only made a single encryption query (N^i, A^i, M^i) and received $C^i || \mathsf{Tag}^i$. It aims to forge using the same nonce but a different associated data A. The adversary must find a correct $\mathsf{Tag} = (\mathsf{Tag}_e + \mathsf{Tag}_a)[n-1, \cdots, n-\tau]$. We consider two subcases: ④a $|A| \neq 0$ and ④b $|A| = 0$.

④a In this case, let's assume that we even provide the adversary with all the functions $\widetilde{R}^{\langle N,x,y \rangle}(.)$, so that the adversary can compute the correct value of Tag_e. Then the adversary's task will reduce to guessing a correct value for the first τ bits of Tag_a. The only relevant information

that the adversary has is the first τ bits of Tag_a^i. We show that even if the whole Tag_a^i is given to the adversary, the chance to correctly guess the first τ bits of Tag_a is still $2^{-\tau}$. This is done by a simple case analysis:

1. if only one of $|A|$ and $|A^i|$ is a multiple of $n+m$ then it is easy to see from Figure 4.12 that the probability to guess the first τ bits of Tag_a is still $2^{-\tau}$;
2. if $a \neq a_i$ then again from Figure 4.12 we can see that the probability to guess the first τ bits of Tag_a is $2^{-\tau}$;
3. otherwise, we have $a = a_i$ and either both $|A|$ and $|A^i|$ are multiple of $n+m$ or neither of them is a multiple of $n+m$. These two cases are similar. Let's consider the first one. As we have $A \neq A^i$ then it must be the case that for some j we have $A_j \neq A_j^i$. So, the j^{th} value XORed to Tag_a , *i.e.* $\widetilde{R}^{\langle j,0\rangle}(A_j)$ is a fresh n-bit random value; hence the adversary's chance to guess the first τ bits of Tag_a is $2^{-\tau}$.

(4b) In this case the adversary has seen $C^i||\mathsf{Tag}^i$, where $\mathsf{Tag}^i = (\mathsf{Tag}_e^i \oplus \mathsf{Tag}_a^i)[n-1,\cdots,n-\tau]$. To get the forged tuple $(N,\varepsilon,C||\mathsf{Tag})$ be accepted and decrypted, it must find the value of $\mathsf{Tag} = \mathsf{Tag}_e[n-1,\cdots,n-\tau]$ (as $\mathsf{Tag}_a = 0^n$ in this case). Now let's give the adversary all functions $\widetilde{R}^{\langle N,x,0\rangle}(.)$ for $1 \leq x \leq \ell$. Even in this case, the adversary has seen no image of the function $\widetilde{R}^{\langle N,x,j\rangle}(.)$ for $j \in \{1,2\}$, since the value $\mathsf{Tag}^i = \mathsf{Tag}_e^i \oplus \mathsf{Tag}_a^i$ that adversary has seen does not reveal any information about Tag_e^i noting that Tag_a^i is random and unrevealed to the adversary. So, the probability that the adversary can correctly guess the first τ bits of $\mathsf{Tag}_e = \widetilde{R}^{\langle N,\ell,j\rangle}(\text{final input})$ for $j = \{1,2\}$ is $2^{-\tau}$. (Note that $j = 1$ when $|C|$ is a multiple of m and $j = 2$ when $|C|$ is not a multiple of m).

(5) $N = N^i$, $A = A^i$, and $|C| = |C^i| = \ell m$ is a multiple of m. We can again ignore all queries except the i^{th} query. Let's assume that we make all functions $\widetilde{R}^{\langle x,y\rangle}$ (for $x \geq 1$ and $y \in \{0,1\}$) used in processing the associate data public to the adversary; i.e assume that the adversary even knows the values of Tag_a and Tag_a^i. Now remember that the adversary must not repeat the known tuple $(N^i, A^i, C^i||\mathsf{Tag}^i)$ as its decryption query, so it must be the case that $C \neq C^i$ as otherwise any $\mathsf{Tag} \neq \mathsf{Tag}^i$ will be incorrect and rejected. Therefore, we may assume that the alleged forgery will be of the form $(N,A,C||\mathsf{Tag})$ such that $C_j \neq C_j^i$ for some $1 \leq j \leq \ell$. Now referring to (the top of) Figure 4.12 it is easy to see that if $C_\ell \neq C_\ell^i$ then the probability that the adversary can correctly guess the value of Tag is $2^{-\tau}$; otherwise there are two cases: (1) if $H_\ell \neq H_\ell^i$ the chance that Tag is correct is $2^{-\tau}$; (2) if the event $H_\ell = H_\ell^i$ happens then adversary can simply use $\mathsf{Tag} = \mathsf{Tag}^i$, but this event only happens with probability at most $\ell 2^{-n}$ noting that $|H_i| = n$ (note that we credit the adversary for any possible collision in the iteration, there are ℓ blocks and the probability of each collision under the random function is 2^{-n}). So, the total success probability in this case is bounded by $\frac{1}{2^\tau} + \frac{\ell}{2^n}$.

(6) $N = N^i$, $A = A^i$, and $|C| = |C^i|$ is not a multiple of m. It is easy to see from Figure 4.12 that the analysis of this case is the same as that of Case 5 and the success probability of the adversary is bounded by $\frac{1}{2^\tau} + \frac{\ell}{2^n}$.

Finally, using the results of Bellare *et al.* [BGM04] we get the bound against adversaries that make q_v decryption (verification) queries as $\frac{q_v}{2^\tau} + \frac{q_v \ell}{2^n}$.

4.3.9 Instantiating Tweakable RFs with PRFs

We proceed to complete the proof of Theorem 4.6 in two steps.

(1) Replace the tweakable RF $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ in $\mathbb{OMD}$ with a tweakable PRF $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$, where $\mathcal{K} = \{0,1\}^k$. The following lemma states the classical bound on the security loss induced by this replacement step. The proof is a straightforward reduction and omitted here.

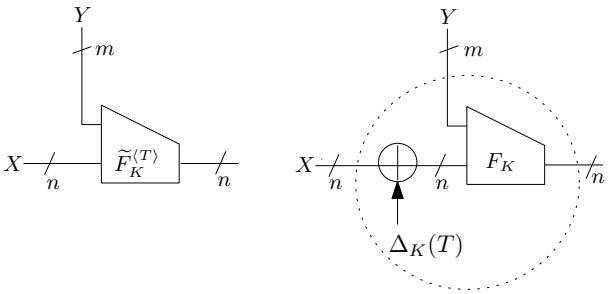


Figure 4.13: Building a tweakable PRF $\widetilde{F}_K^{\langle T\rangle} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ using a PRF $F_K : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. There are several efficient ways to define the masking function $\Delta(T)$ [Rog04a, CS08, KR11]. We use the method of [KR11].

Lemma 4.8 *Let $\widetilde{R} : \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a tweakable RF and $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a tweakable PRF. Then*

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{priv}}_{\mathbb{OMD}[\widetilde{F},\tau]}(t, q_e, \sigma_e, \ell_{\max}) &\leq \mathrm{Adv}^{\mathrm{priv}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e, \sigma_e, \ell_{\max}) + \mathrm{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t', \sigma_e) \\
\mathrm{Adv}^{\mathrm{auth}}_{\mathbb{OMD}[\widetilde{F},\tau]}(t, q_e, q_v, \sigma, \ell_{\max}) &\leq \mathrm{Adv}^{\mathrm{auth}}_{\mathbb{OMD}[\widetilde{R},\tau]}(q_e, q_v, \sigma, \ell_{\max}) + \mathrm{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t'', \sigma)
\end{aligned}$$

where q_e and q_v are, respectively, the number of encryption and decryption queries, $q = q_e + q_v$, $\ell_{\max}$ denotes the maximum number of m-bit blocks in an encryption or decryption query, $t' = t + cn\sigma_e$ and $t'' = t + c'n\sigma$ for some constants c, c', and σ_e and σ are the total number of calls to the underlying compression function F in all queries asked by the CPA *and* CCA *adversaries against the privacy and authenticity of the scheme, respectively.*

② We instantiate a tweakable PRF using a PRF by means of XORing (part of) the input by a mask generated as a function of the key and tweak as shown in Fig. 4.13. This method to tweak a PRF is (essentially) the XE method of [Rog04a]. In OMD the tweaks are of the form $T = (\alpha, i, j)$ where $\alpha \in \mathcal{N} \cup \{\varepsilon\}$, $1 \leq i \leq 2^{n-8}$ and $j \in \{0, 1, 2\}$. We note that not all combinations are used; for example, if $\alpha = \varepsilon$ (empty) which corresponds to processing of the associate data in Figure 4.11 then $j \neq 2$. The masking function $\Delta_K(T) = \Delta_K(\alpha, i, j)$ outputs an n-bit mask such that the following two properties hold for any fixed string $H \in \{0,1\}^n$:

1. $\Pr[\Delta_K(\alpha, i, j) = H] \leq 2^{-n}$ for any (α, i, j)
2. $\Pr[\Delta_K(\alpha, i, j) \oplus \Delta_K(\alpha', i', j') = H] \leq 2^{-n}$ for $(\alpha, i, j) \neq (\alpha', i', j')$

where the probabilities are taken over random selection of the secret key K.

It is easy to verify that these two properties are satisfied by the specific masking scheme of OMD as described in Section 4.3.4.

Lemma 4.9 *Let $F : \mathcal{K} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be a function family with key space $\mathcal{K}$. Let $\widetilde{F} : \mathcal{K} \times \mathcal{T} \times (\{0,1\}^n \times \{0,1\}^m) \to \{0,1\}^n$ be defined by $\widetilde{F}_K^{\langle T\rangle}(X||Y) = F_K((X \oplus \Delta(T))||Y)$ for every $T \in \mathcal{T}, K \in \mathcal{K}$, $X \in \{0,1\}^n, Y \in \{0,1\}^m$ and $\Delta_K(T)$ is the masking function of OMD as defined in Section 4.3.4. If F is PRF then $\widetilde{F}$ is tweakable PRF. More precisely*

$$\mathrm{Adv}^{\widetilde{\mathrm{prf}}}_{\widetilde{F}}(t, q) \leq \mathrm{Adv}^{\mathrm{prf}}_{F}(t', 2q) + \frac{3q^2}{2^n}\,.$$

The proof is a simple adaptation of a similar result on the security of the XE construction (to tweak a blockcipher) in [KR11]. As we use a PRF rather than PRP, our bound has two main terms. The first term is a single birthday bound loss of $\frac{0.5q^2}{2^n}$ to take care of the case that a collision might happen when computing the initial mask $\Delta_{N,0,0} = F_K(N||10^{n-1-|N|}, 0^m)$ using a PRF (F) rather than a PRP (as in [KR11]). The analysis of the remaining term (*i.e.* $\frac{2.5q^2}{2^n}$) is essentially the same as the similar part in [KR11], but we note that in the context of our construction as we are directly dealing with PRFs

unlike [KR11] in which PRPs are used, the bound obtained here does not have any loss terms caused by the switching (PRF/PRP) lemma. Therefore, instead of the $\frac{6q^2}{2^n}$ bound in [KR11] (from which $\frac{3.5q^2}{2^n}$ is due to using the switching lemma) our bound has only $\frac{2.5q^2}{2^n}$.

4.3.10 In Summary: Features of OMD

Based on a Single Primitive, Provable Secure and Requiring Minimal Operations. OMD is designed as a mode of operation for a keyed compression function. Together with blockciphers and permutations, compression functions are among the most well-known and widely used symmetric key primitives. We have a rich source of secure compression functions thanks to more than two decades of public research and standardization activities on hash functions.

The security goals of privacy and authenticity for OMD are achieved provably in the sense of reduction-based cryptography. That is, any attack against these security goals will imply an attack against the classical PRF property of the underlying compression function. We note that any keyed compression function (either a dedicated-key one or keyed via some part of its input) must provide the classical PRF property when its key is secret as otherwise it will be considered useless for any secret key application, *e.g.* for being used as a MAC. That is, the base PRF assumption on the compression function upon which the security of OMD relies is highly assured for compression functions of the practical, standard hash functions, thanks to the vast amount of cryptanalytic work on these functions.

The only operations that OMD needs *in addition to* its core compression function are the basic operations of bitwise XORing two binary strings and shifting a binary string.

Integrated (One-Pass) AEAD Scheme. In OMD the mechanisms for providing privacy and authenticity of the message are coupled in a single pass of (a variant of) the Merkle-Damgård iteration of the compression function. This is aimed to make OMD as much efficient as possible (up to the limits that are inherent to any compression function based AEAD scheme).

On-line Encryption and Internally On-line Decryption. OMD encryption is on-line. That is, it outputs a stream of ciphertext as a stream of plaintext arrives with a constant latency and using constant memory. After receiving an indication that the plaintext is over, the final part of ciphertext together with the tag is output.

OMD decryption is *internally* on-line: one can generate a stream of plaintext bits as the stream of ciphertext bits comes in, but no part of the plaintext stream will be revealed before the whole ciphertext stream is decrypted and the tag is verified to be correct. That is, nothing about the decrypted plaintext should be made available to adversaries if the tag is incorrect signifying that the queried ciphertext is invalid. This feature is closely related to the secrecy against RUP presented by Andreeva *et al.* [And14].

Flexibility of the Key Size and Efficiency. OMD-sha256 can support any key length between 80 bits and 256 bits. This will be useful for applications requiring unconventional key lengths, *e.g.* 96-bit keys.

If implemented with a member of the SHA family, OMD can take advantage of the newly introduced Intel instructions that support performance acceleration of the Secure Hash Algorithm (SHA) on Intel Architecture processors. In particular, our main recommended scheme, called OMD-sha256, is aimed to get the most out of these new performance accelerating instructions.

Resistance Against Software-Level Timing Attacks. Most AES software implementations risk leaking their keys through cache timing [Ber05] unless they are implemented on machines with Intel CPUs supporting the constant-time AES-NI and `PCLMULQDQ` instructions. In comparison, we note that the only operations in OMD-sha256 are: bitwise XOR, AND and OR of two binary strings (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), fixed-distance (left and right) shift of a binary string (32-bit words in the compression function of SHA-256 and 256-bit words in the OMD iteration), and 32-bit addition (of words in the compression function of SHA-256). These

operations have the virtue of taking constant time on typical CPUs in which case the implementations can avoid software-level timing based side-channel leaks.

4.3.11 Further Developments

4.3.11.1 Pure OMD (p-OMD)

After we proposed the initial design of the OMD scheme, Reyhanitabar, Vaudenay and Vizár [RVV15] presented pure OMD (p-OMD) as a new version. p-OMD inherits the security features of OMD, providing higher efficiency. Compared to OMD, p-OMD removes the XOR MAC algorithm and is only based on the MD iteration. For a message of ℓ blocks and associated data of a blocks, OMD needs $\ell + a + 2$ calls to the compression function while p-OMD only requires max $\{\ell, a\}$ + 2 calls. In the usual case where $\ell \geq a$, p-OMD makes just $\ell + 2$ calls to the compression function.

4.3.11.2 Getting the Tag as a By-Product of Encryption

It might be interesting to see if the authentication of a single block can be obtained as a by-product of encryption. The study of the following ideas could be appealing.

Consider a block cipher, *e.g.* AES-128 and the construction depicted in Figure 4.15. The idea consists in using an inexpensive one-way function for computing the Tag from an intermediate encryption state. The lightweight perspective offered by such a function could provide tags almost for free (and, thus, authentication).

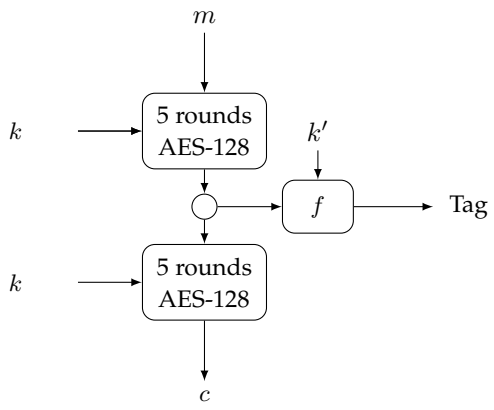


Figure 4.14: Authentication as a by-product of encryption.

Example 4.3 *The lightweight function f might be defined as*

$$f(x) = (x^2 \bmod K) \bmod 2^{128},$$

where K is a 300-bit secret prime.

Conceptually, the previous idea consists in allowing the leakage of certain bits to achieve authentication.

In other words, we create a deliberate side channel. As today the research community has extensively researched side channel attacks of why not try and apply this knowledge to obtain authentication and confidentiality by combining "deliberate side channel attacks" with encryption?

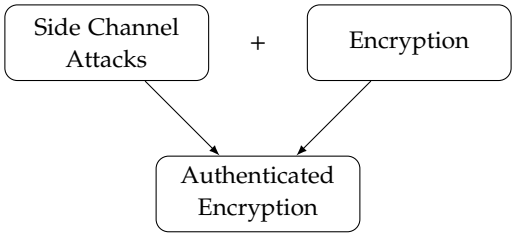


Figure 4.15: Authenticated encryption: a new perspective.

Algorithm 10: $\mathcal{D}_K(N, A, \mathbb{C})$

if $|N| > n - 1$ or $|\mathbb{C}| < \tau$ **then**
  return $\perp$
end if
$C_1||C_2||\cdots||C_{\ell-1}||C_\ell||\mathsf{Tag} \xleftarrow{m} \mathbb{C}$, where $|C_i| = m$ for $1 \le i \le \ell - 1$, $|C_\ell| \le m$ and $|\mathsf{Tag}| = \tau$
$\Delta \leftarrow F_K(N||10^{n-1-|N|}, 0^m)$ ▷ initialize $\Delta_{N,0,0}$
$H \leftarrow 0^n$
$\Delta \leftarrow \Delta \oplus L[0]$ ▷ compute $\Delta_{N,1,0}$
$H \leftarrow F_K(H \oplus \Delta, \langle\tau\rangle_m)$
for $i \leftarrow 1$ to $\ell - 1$ **do**
  $M_i \leftarrow H \oplus C_i$
  $\Delta \leftarrow \Delta \oplus L[\mathtt{ntz}(i+1)]$
  $H \leftarrow F_K(H \oplus \Delta, M_i)$
end for
$M_\ell \leftarrow H \oplus C_\ell$
if $|C_\ell| = m$ **then**
  $\Delta \leftarrow \Delta \oplus 2.L_*$
  $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell)$
end if
else
  if $|C_\ell| \neq 0$ **then**
    $\Delta \leftarrow \Delta \oplus 3.L_*$
    $\mathsf{Tag}_e \leftarrow F_K(H \oplus \Delta, M_\ell||10^{m-|M_\ell|-1})$
  end if
end if
else
  $\mathsf{Tag}_e \leftarrow H$
end if
$\mathsf{Tag}_a \leftarrow \textsc{Hash}_K(A)$
$\mathsf{Tag}' \leftarrow (\mathsf{Tag}_e \oplus \mathsf{Tag}_a)[n-1, \cdots, n-\tau]$
if $\mathsf{Tag}' = \mathsf{Tag}$ **then**
  return $M \leftarrow M_1||M_2||\cdots||M_\ell$
end if
else
  return $\perp$
end if

Chapter 5

Algorithms for Embedded Cryptography

Lots of people working in cryptography have no deep concern with real application issues. They are trying to discover things clever enough to write papers about.
Whitfield Diffie.

Summary

This chapter is dedicated to the new but increasingly active field of *lightweight cryptography*. Lightweight cryptography has emerged as a direct consequence of the increasing need for small, smart devices. We stress that associating the terms *cryptography* and *lightweight* does not mean weak systems, but cryptographic solutions suitable for highly resource-constrained mobile devices.

Lightweight (dedicated) solutions for RFID tags are presented in Section 5.1.1. Possible risks are examined and different measures taken for improving RFID's security and privacy levels are discussed in the same section.

Section 5.4 describes a new backtracking-based multiplication algorithm, especially suited for lightweight microprocessors when one of the operands is known in advance. The constant operand is encoded in a computation-friendly way, based on linear relationships amongst its sub-words. Our result is backed-up by an implementation on a 68HC05 microprocessor showing that the new algorithm indeed yields a twofold speed improvement over classical multiplication for 128-byte numbers.

Barrett's modular reduction algorithm is described in Section 5.2.1. Section 5.2 presents a method allowing to double the speed of Barrett's algorithm by using specific composite moduli. We show how to generate such Barrett-friendly RSA moduli and apply the idea to other cryptographic primitives (*e.g.* DSA) - where we find instances whose entire collection of parameters is multiplication-friendly.

Section 5.3.1 presents polynomial extensions of Barrett's modular reduction algorithm. We devise new BCH speed-up strategies using Barrett's polynomial reduction method as well as (optimised) Linear Feedback Shift Register (LFSR) architectures, providing comparisons between the proposed solutions. Section 5.3.2 describes a new error-correcting code (ECC) inspired by the Naccache-Stern cryptosystem. This ECC happens to be more efficient than some established ECCs for certain sets of parameters. Section 5.5 proposes a new building block called *Pace Regulator*. The *Pace Regulator* is inserted between the randomness consumer and a von Neumann extractor to streamline the pace of random bits.

Section 5.6.1.1 provides an overview and a classification of fault attacks, shortly describing possible countermeasures. Following the work of Naccache, Smart and Stern [NSS04], Section 5.6 proposes fault attacks on elliptic curve cryptography implementations.

5.1 Lightweight Cryptography for Embedded Devices

The relatively new and continuously developing sub-field, *Lightweight Cryptography*, can be defined as a collection of cryptographic primitives, techniques and ciphers with suitable implementations in highly resource-constrained mobile devices.

Lightweight Cryptography is hence at a crossroad between cryptography, computer science and electrical engineering. It focuses on new designs, adaptations or efficient implementations of cryptographic primitives and protocols.

Considering constraints and the fact that the embedded devices operate in hostile environments (note that we have to take into consideration physical attacks too), there is an increasing need for security solutions, especially constructed in view of the current ubiquitous computing tendency.

There are three categories of solutions for providing cryptographic primitives for lightweight applications: optimized low-cost implementations for standardized and well-known algorithms, slightly modified, well investigated ciphers and new ciphers especially designed to meet low hardware implementation costs. Thus, the notion of *Lightweight Cryptography* is frequently used to describe optimizations of existing primitives as well as new designs.

The growth of ubiquitous computing made security and privacy increasingly important [Pos09]. Traditional cryptographic algorithms are unsuited for constrained devices. The main issue when addressing this concern is how to reach sufficient security using only little computing power. Hence, the trade-off between lightweightness and the security is the cornerstone of lightweight cryptography.

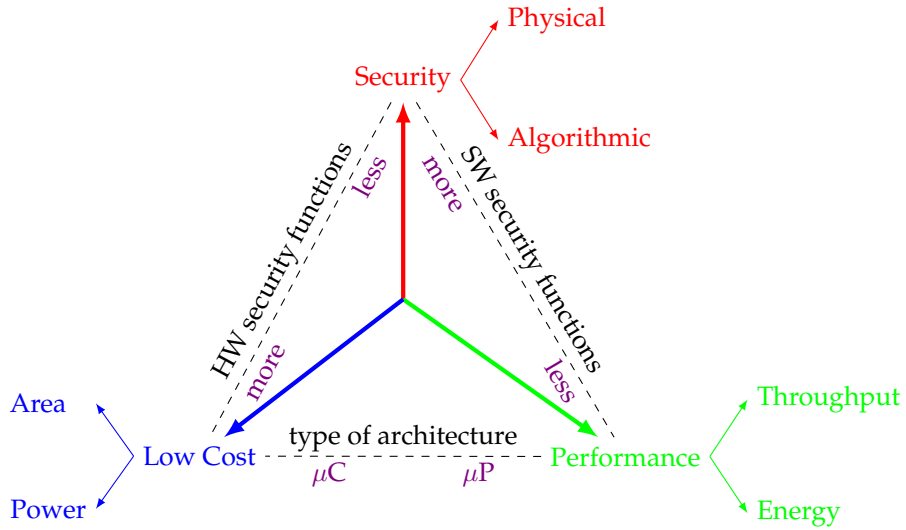


Figure 5.1: Securing Devices

Our research focused on cryptography for limited devices for which trade-offs between performance, security, and cost are highly important. The constraints are computing power, memory, bandwidth, or vulnerability to attacks. Two of the most common examples of lightweight devices are further studies: RFID tags and wireless sensor networks. Available implementations of already established cryptographic algorithms and libraries especially developed for constrained devices are very far from being optimal. As an example, there do not exist many cryptographic libraries engineered for frugality. Many of these functions are prone to algorithmic optimizations of the types proposed in this chapter.

The Internet of Things (IoT) consists in spatially distributed nodes forming a network, able to control or monitor physical or environmental conditions (such as temperature, pressure, image and sound), perform computations or store data.

IoT nodes are typically low-cost devices with limited computational resources and limited battery. They transmit the data that they acquire through the network to a gateway, also called the *transceiver*. The

gateway collects information and sends it to a processing unit. Nodes are usually deployed in hostile environments, and are therefore susceptible to physical attacks, harsh weather and communication interferences.

Lightweight Block Ciphers. A suitable solution for lightweight cryptography, other than efficiently implementing or slightly modifying an already trusted cipher, is to design a new optimized ciphers a fresh. Among such solutions, we mention PRESENT, which is a 31 round substitution-permutation network based block cipher (and having a 64 bits block size, and a key of 80 or 128 bits). Poschman stressed in his thesis [Pos09] that: "*The main design philosophy during the design of* PRESENT *was simplicity: no part of the cipher was added without a good reason for it, like thwarting an attack*".

PRESENT, KATAN/KTANTAN, ITUbee and PRIDE are some common lightweight block-ciphers. We present each of them briefly in the next paragraphs.

We stress that PRESENT was not inspired by AES, even though many SPN based ciphers have a structure close to AES. PRESENT is a typical algorithm designed especially for hardware, using simple wiring and bit-oriented permutations. PRESENT has been standardized under the reference ISO/IEC 29192 [iso12].

KATAN and KTANTAN [DCDK09] are a family of small hardware-oriented block ciphers. The optimization of the physical footprint is at the core of these two designs. The main difference between KATAN and KTANTAN is the key schedule, as KTANTAN uses a fixed hardware encoded key that cannot be changed.

ITUbee is a software oriented cipher using an internal Feistel structure with key whitening at the beginning and end of the encryption. Its main features are low power consumption and low memory management in software.

PRIDE is a block cipher that focuses on the design of the linear layer in Substitution-Permutation Networks and its main target is 8 bit microcontrollers. The model implementation has been optimized to the AVR instruction set.

Lightweight Pseudo-Random Number Generators. Poschman [Pos09] applies PRESENT's hardware efficiency to seed public-key cryptography: in output feedback mode (OFB) and, hence, making it a stream cipher. This stream cipher is used as a pseudo-random number generator for a public-key identification scheme.

General Aspects. Saarinen and Engels [SE12] analysed in depth the constraints and sketched the guidelines of a successful lightweight cryptographic primitive design.

Targeting an RFID or lightweight sensor network "Do It All Cipher" (DIAC) for IoT, their conclusion revolves around the following observations.

- The latency of the primitive has to be less than 50 cycles and encryption and decryption throughputs must exceed more than a bit per cycle.
- Power usage should be less than 1 to 10μW/MHz on average with peaks below 3μW and 30μW respectively.
- The primitive must be able to operate without significant modification as a single-pass tweakable authenticated encryption and decryption algorithm as well as a cryptographically secure hash.
- The padding scheme and the operations taken into account for various inputs, including initialization vectors and authenticated associated data must be clearly stated. Additional data size must be small to avoid message expansion.
- The initialization vector does not have to be a nonce (one must target security in a repeated chosen-IV attack). Security level must be high, *i.e.* with key and state sizes considered under all attack models.
- Hardware implementation must not consist of more than 2000 GE, including state memory and both encryption and decryption functions. Software implementation speed and size across all MCU and CPU platforms has to be taken into consideration as well.

From a software perspective, the time complexity of a primitive comprises two ideas which are discussed next. The number of clock cycles necessary to process a data byte determines the speed of the algorithm, but this is not enough. Hence, achieving a high speed at the cost of a high computational load, for

instance because of a complicated key schedule, may not be acceptable in some cases. Latency, *i.e.* the delay given by a high computational load (considered in clock cycles) must be considered as well.

Complexity, in this case, actually refers to memory complexity, *i.e.* the amount of RAM necessary to fulfil the computations. Nonetheless, *e.g.* in a flash memory, the space required to store the algorithm must be taken into account.

Now, from a hardware perspective, the memory requirement corresponds to the number of logical gates necessary to implement the primitive. This quantity is measured using a unit called G(ate) E(quivalence), where one GE corresponds to the area of a NAND gate.

Time efficiency (throughput) is measured in bits per second at a given clock frequency, usually 100 Hz, which corresponds to the amount of data processed in one second using the given clock frequency. Latency must be considered in hardware, as latency is correlated with the time needed to *e.g.* derive the sub-keys. Nevertheless, giving an estimation of these quantities is rather complex.

Measuring the lowest necessary resources for lightweight cryptography applications, we will not obtain a connection between performance in software and hardware. In reality, between the two implementation categories arise design trade-offs. We may consider as an example the use of S-boxes, which can be very efficient when implemented in software (*e.g.* using a look-up table). But, from a hardware perspective, they are not so easy to implement. This is one of the reasons why, *e.g.*, Piccolo [SIH+11] was designed implementing the S-box based on a small Feistel Network.

There is always the other side of the coin, thus, bit-oriented permutations used in PRESENT for example, can be implemented in hardware for free using wires. Nevertheless, such an approach is rather slow in software.

Based on the above, we may conclude that it is hard to aim at both software and hardware efficiencies when it comes to lightweight cryptography.

5.1.1 RFID Tags: A Cryptography Perspective

RFID tags were initially developed during the second World War [Lan]. Nevertheless, they continue playing an important role given the requirement of easy data sharing backed up by fast and secure communication channels. Their remarkable development found motivation in, *e.g.*, the wide adoption of biometric passports [1] as well as the need of replacing bar codes. RFID tags imply simplicity in providing authentication. Such devices communicate with RFID readers, which are connected to a back-end database server through a permanent link.

An RFID tag consists of ① an *antenna* that emits and receives radio signals and ② a *chip* incorporating a modulator and demodulator for signal processing and a circuit for memory, information processing, and possibly functions dedicated to specific tasks such as measuring a physical phenomenon's scale. RFID tags do not require physical contact to communicate. Compared to smartcards, RFID tags may be much smaller, ranging between hundreds of μm^2 to a few cm^2.

There are three classes of tags:

① *Passive* - these represent the vast majority of tags, having no battery (being powered by a magnetic resonance induction field);

② *Active* - including a battery, they operate autonomously; being costly, such tags are reserved to high-end applications (*e.g.* the US Department of Defence's inventory-tracking system);

③ *Semi-passive* - they carry a battery used exclusively for internal computation and rely on an external power source for communication.

Conventional cryptographic solutions cannot be implemented on such constrained devices. A similar situation took place towards the end of the 1980s, when smart cards emerged. Cryptographers made smart cards secure by employing more efficient cryptographic primitives and incorporating powerful control units in the cards. But even if the constraints are similar, the current situation is different. The limiting factor now isn't technology but cost. So, even if implementing sophisticated cryptographic

1. In June 2014 [OK14] estimated the existence of 500 million biometric passports issued by more than 100 countries (of which 485 million were already in circulation).

protocols in a tiny chip is technically possible, for low-end RFID tags, the circuit dedicated to security can't exceed a certain area.

Low-end RFID tags' chips are limited to 10,000 GEs, of which only 2,000 GEs are for security.

Four main approaches developed in parallel have been considered as suitable solutions: ① efficiently implementing established ciphers, ② choosing established ciphers with smaller parameters [2], ③ designing new ciphers, and ④ developing fully dedicated solutions. Variants of well-known cryptographic algorithms have been implemented. One example is a serialized version of DES that requires 2,310 GEs and encrypts a plaintext within 144 clock cycles. Another is a 3,100 GE version of AES in an encryption-only architecture. A relevant conclusion in this area was that the NTRU (n-th degree TRUncated polynomial ring) cryptosystem shows promise in this area.

DES was also the target of several proposals to fit a classic cryptographic primitive in a low-end RFID tag. Two examples are DESL (DES Lightweight) and DESXL (DES Extra-Lightweight), versions of DES with fewer rounds and simplified building blocks. In addition, public-key-based solutions have been investigated, focusing on standardized elliptic curves, although fitting these solutions to small chips has been considered impossible. At the same time, researchers have proposed symmetric algorithms such as Hight, Clefia, Squash, and PRESENT, with limited success. Cipher designers have also proposed innovative ciphers, such as KATAN and LED (Light Encryption Device), with duplicated internal components and shorter keys and block sizes. However, this approach has turned out to be risky, as numerous proposals were quickly shown to be insecure. One of the main dedicated solutions developed for RFID technology is discussed next.

5.1.1.1 HB Protocols

Classic cryptographic primitives may be unsuitable for RFID tags. An alternative is cryptography based on $\mathcal{NP}$-complete problems. This area has seen much research since the 1970s. These endeavours laid the foundations of public-key cryptography and to the development of cryptosystems with "easily quantifiable" computational security. Unfortunately, the vast majority of $\mathcal{NP}$-complete based schemes were broken. So, attention moved back to problems which are not proven to be $\mathcal{NP}$-complete, such as factoring and discrete logarithms. However, the past years have seen several innovative protocol proposals based on the LPN (Learning Parity with Noise) problem [BKW03] and its generalization, the LWE (Learning With Errors) problem [Reg10]. Both problems are $\mathcal{NP}$-complete. In particular, researchers used these problems to design the HB (Hooper and Blum) family of efficient authentication protocols [HB01, JW05, GRS08]. HB protocols are referred to as lightweight and even ultralightweight protocols [3] in the literature [JW05, GRS08, Kho14]. However, implementation results are not detailed (*e.g.* the required number of GEs). Our experimental results have shown that even the HB protocol would indeed fit in a device with 3000-8000 GEs. Nonetheless, we consider that some of its variants, *e.g.* hHB, would require more GEs [4]. While hHB is provably secure in a stronger model, there is a need for optimizing its implementation. Thus, we consider HB protocols interesting proposals at least from a theoretical perspective.

Definition 5.1 (The LPN Problem) *Let x be a binary vector of length k and η a real positive number smaller than $1/2$ (typically, η is equal to $1/8$ or $1/4$). Consider an oracle that, at each invocation*
– *generates a random k-bit vector a and a bit b set to 1 with probability η*
and
– *returns $a \cdot x \oplus b$.*
The LPN problem is to recover the secret vector x given access to this oracle.

In the following, Ber_ε denotes the Bernoulli distribution with parameter ε.

HB Protocol. HB is a "pen-and-paper" protocol, requiring very simple operations. For each authentication both parties compute a bit as the scalar product of two binary vectors, and the prover flips its output with probability ν to mimic the LPN's noise. The procedure is repeated r times. Authentication succeeds

2. *e.g.* block size, key length, internal state of the algorithm
3. particularly suitable for RFID tags
4. even though the authors claim that the "computational and storage capabilities of the RFID tag" are not exceeded

if some minimal number of rounds r succeed. This protocol is secure only against passive eavesdroppers. An HB protocol round is presented in Figure 5.2. When the noise

The HB protocol consists of $r = r(k)$ authentication rounds between the reader and the tag.

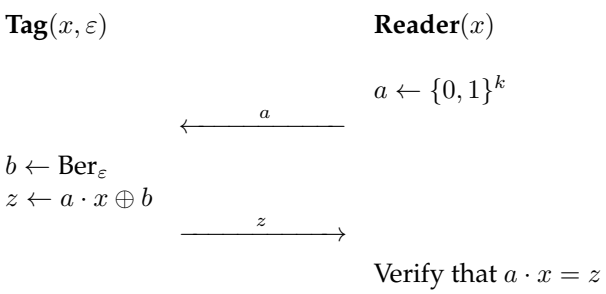


Figure 5.2: A round of the HB protocol, where x is a binary vector of length k, α is a k-bit vector, b is a bit initially set to 1 and Ber_ε denotes the Bernoulli distribution with parameter ε.

As the noise increases, false rejection [5] will grow. Relations between the level of noise and false rejection rate were analysed by Katz *et al.* in [KSS10].

HB$^+$ Protocol. Juels and Weis suggested adapting HB for RFID authentication [JW05]. They proposed HB$^+$, which is an HB variant secure against active adversaries (*i.e.* assuming that the adversary can directly interact with tags and readers, although not concurrently). HB$^+$ becomes insecure when the adversary has concurrent access to both parties, as the GRS (Gilbert, Robshaw, and Sibert) attack showed [GRS05].

The HB$^+$ protocol consists of $r = r(k)$ authentication rounds between the reader and the tag. Two random secret keys x and y of length k are shared between the parties. An HB protocol round is presented in Figure 5.3.

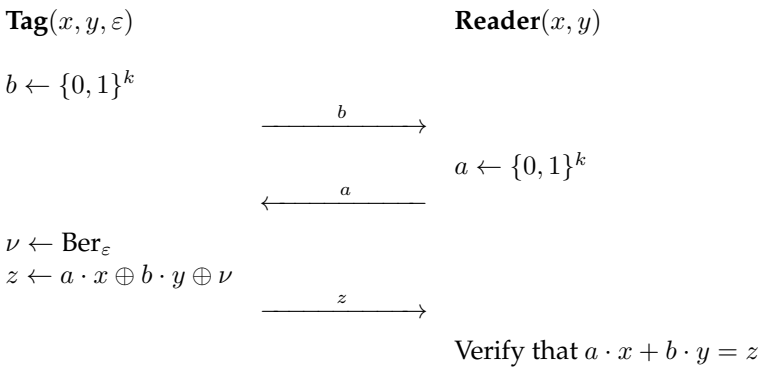


Figure 5.3: A round of the HB$^+$ protocol, where x and y are binary vectors of length k, b and α are k-bit vectors and Ber_ε denotes the Bernoulli distribution with parameter ε.

HB$^\#$ Protocol. After a series of attempts to fix HB$^+$ and render it immune to the GRS attack, Henri Gilbert and his colleagues proposed HB$^\#$ [GRS08]. They proved that HB# is secure against adversaries similar to the ones considered by the GRS attack, as long as the LPN problem is hard. In addition, the authors argued that HB$^\#$ was secure against general man-in-the-middle adversaries. This conjecture eventually turned out to be false, as the OOV (Ouafi, Overbeck, and Vaudenay) attack on HB$^\#$ showed [OOV08].

Let $X \in \mathbb{F}_2^{k_x \times m}$ and $Y \in \mathbb{F}_2^{k_y \times m}$ be Toeplitz matrices.

5. rejection of the honest prover

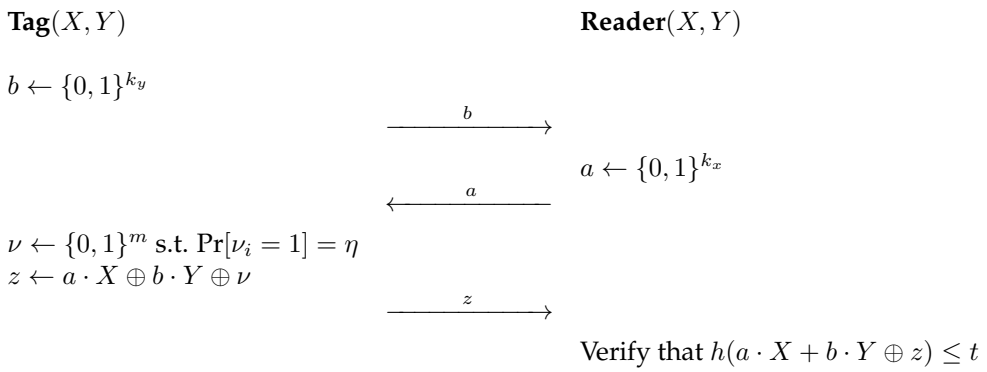


Figure 5.4: A round of the HB$^{\#}$ protocol.

hHB. Given the attacks on HB$^{\#}$ various proposals of fixed variants appeared. The hHB protocol stands for harder HB^{+}. The core idea of it is to let the reader choose a random k-bit secret x and then to send it to the tag securely. Hence, hHB consists of two stages. In the first stage the reader selects a random secret x and transmits it to the tag. The second stage of hHB is identical to the HB^{+} protocol.

The function f_s changes the order of elements in a triplet. For a complete definition of hHB, we refer the reader to [Kho14]. The hHB protocol is presented in Figure 5.5.

The hHB protocol was designed to provide resistance to general Man-in-the-Middle adversaries. The intuition behind this protocol is the belief that HB^{+} protocol's weakness stems from not changing the secrets x and y. Thus, a random number Γ generated by the reader replaces the two secrets.

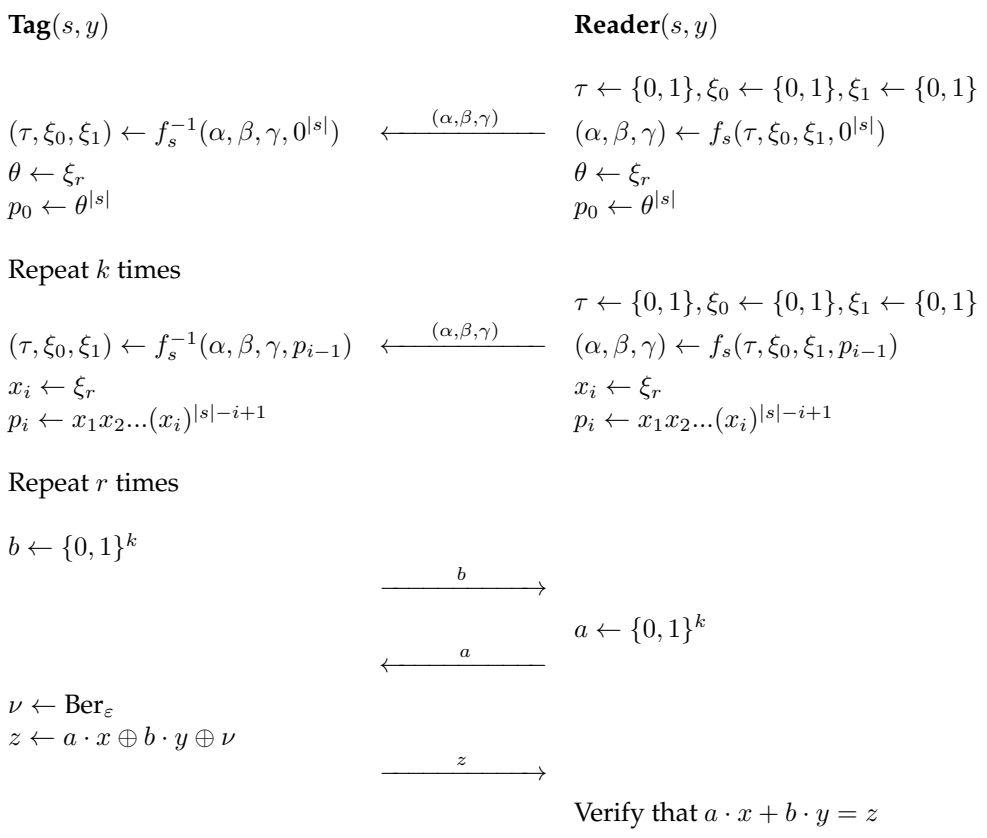


Figure 5.5: The hHB protocol.

5.2 Double-Speed Barrett Moduli

We will now present a method allowing to double the speed of Barrett's algorithm by using specific composite moduli. As already addressed in Section 5, computational speed-ups are particularly useful for lightweight devices where such optimizations can make a difference in terms of power consumption, cost and processing time. The generation of composite moduli with a predetermined portion is a well-known technique [Joy08, Len98, VZ95] and the use of such moduli is considered, *in statu scientiæ*, as safe as using randomly generated composite moduli.

Related Work: Quisquater [Qui92] proposes a fast RSA encryption implementation for which the modular exponentiation step is divided into two successive operations allowing quasi-reduction based on a predetermined multiple of the modulus. A fast exponentiation method modulo a prime number for discrete logarithm based systems is presented by van Tilbourg in [vT91]. Douguet and Dupaquis [DD08] describe a modified Barrett modular reduction algorithm whose purpose is the acceleration of this type of operation in certain (elliptic curve) groups of known moduli. Thus, the approach they consider implies moduli with a given form, *e.g.* the recommended ones from [fip00]. Estimations of the speed-ups are not provided, but the resistance of various architectures to different physical attacks is discussed. A general form of the Barrett constant and of the quotients (when certain moduli are used) are described. As an example of the proposed techniques, the Elliptic Curve Digital Signature Algorithm (ECDSA) [fip13] is taken into account.

We stress that no specific modulus generation algorithm is presented in [DD08]. The approach of [DD08] is rather a practical one, whereas our goal is to provide formal mathematical models for moduli with a predetermined portion generation.

Knežević, Batina and Verbauwhede [KBV09] propose two sets of moduli for which Barrett's modular reduction algorithm can be implemented by avoiding the pre-computation of the Barrett constant. The types of moduli considered throughout this section do not fall into those sets.

Organisation: Section 5.2.2 recalls background concerning composite moduli a predetermined portion. Section 5.2.3 introduces the core idea, that leverages Section 5.2.2 to generate Barrett-friendly RSA moduli. In Section 5.2.4, we apply this idea to other cryptographic primitives, such as DSA [fip13].

5.2.1 Barrett's Reduction Algorithm

Modular multiplication and modular reduction are the atomic constituents of most public-key cryptosystems. Amongst the numerous algorithms for performing these operations (*e.g.* [BGV94, Bri83, Knu81, Mon85]), a particularly elegant method was proposed by Barrett in [Bar87]. This method assembles the operation $a \bmod b$ from bit shifts, multiplications and additions in $\mathbb{Z}$. This allows to build modular reduction at very marginal code or silicon costs by leveraging existing hardware or software multipliers. For a very detailed comparison of the principal modular reduction strategies, we refer the reader to [BGV94].

Notations. For a given a, let $\|a\| = 1 + \lfloor \log_2 a \rfloor = \lceil \log_2 (a+1) \rceil$. That is, $\|a\|$ will denote the bit-length of a throughout the next two sections. $a|b$ will represent the concatenation of the bit-strings a and b.

$x \gg y$ will denote binary shift-to-the-right of x by y places *i.e.*:

$$x \gg y = \left\lfloor \frac{x}{2^y} \right\rfloor .$$

Barrett's algorithm (Algorithm 11) approximates the result $c = d \bmod n$ by a quasi-reduced number $c + \epsilon n$ where $0 \leq \epsilon \leq 2$. We denote $N = \|n\|, D = \|d\|$ and set a *maximal bit-length reduction capacity* L such that $N \leq D \leq L$. The algorithm will function as long as $D \leq L$. In most implementations $D = L = 2N$. The algorithm uses the pre-computed constant $\kappa = \lfloor 2^L / n \rfloor$ that depends only on n and L. The reader is referred to [Bar87] for a proof and a thorough analysis of this algorithm.

Algorithm 11: Barrett's Algorithm

Input: $n < 2^N, d < 2^D, \kappa = \left\lfloor \frac{2^L}{n} \right\rfloor$ where $N \leq D \leq L$

Output: $c = d \bmod n$

1 $c_1 \leftarrow d \gg (N-1)$

2 $c_2 \leftarrow c_1 \kappa$

3 $c_3 \leftarrow c_2 \gg (L-N+1)$

4 $c_4 \leftarrow d - nc_3$

5 **while** $c_4 \geq n$ **do**

6 $\quad c_4 \leftarrow c_4 - n$

7 **end while**

8 **return** c_4

Example 5.1 *Our goal is to reduce* 8619 *mod* 93. *The correct result should be* 63.

$n = 93 \;\Rightarrow\; N = 7$

$\kappa = \left\lfloor \frac{2^{32}}{n} \right\rfloor$ =10110000001011000000101100

$d = 8619$ =10000110101011

c_1 =1000011~~0101011~~ = 10000110

c_2 =10111000011011100001101110000100 0

c_3 =1011100 ~~00110111000011011100001000~~ =1011100

nc_3 =10000101101100

c_4 = 63

Work Factor: $\|c_1\| = D - N + 1 \simeq D - N$ and $\|\kappa\| = L - N$ hence their product requires $w = (D-N)(L-N)$ elementary operations. $\|c_3\| = (D-N) + (L-N) - (L-N+1) = D-N-1 \simeq D-N$. The product nc_3 will therefore claim $w' = (D-N)N$ elementary operations. All in all, work amounts to $w + w' = (D-N)(L-N) + (D-N)N = (D-N)L$.

The results presented in Section 5.2 lead to the halving of this work factor.

5.2.1.1 Dynamic Constant Scaling

Lemma 5.1 *If* $U \leq L$, *then* $\bar{\kappa} = \kappa \gg U = \left\lfloor \dfrac{2^{L-U}}{n} \right\rfloor$.

Proof: $\exists\, \alpha < 2^U$ and $\beta < n$ (integers) verifying: $\bar{\kappa} = \dfrac{\kappa}{2^U} - \dfrac{\alpha}{2^U}$ and $\kappa = \dfrac{2^L}{n} - \dfrac{\beta}{n}$.

Therefore,

$$\min_{\alpha\,\beta}\left(\frac{2^{L-U}}{n} - \frac{\beta+\alpha n}{2^U n}\right) \leq \bar{\kappa} = \frac{2^{L-U}}{n} - \frac{\beta+\alpha n}{2^U n} \leq \max_{\alpha,\beta}\left(\frac{2^{L-U}}{n} - \frac{\beta+\alpha n}{2^U n}\right)$$

and finally,

$$\frac{2^{L-U}}{n} - 1 < \frac{2^{L-U}}{n} - 1 + \frac{1}{2^U n} \leq \bar{\kappa} \leq \frac{2^{L-U}}{n}\,.$$

□

Work factor: We know now that $\bar{\kappa} = \kappa \gg L - D$. Let $c_5 = D - N + 1$. Replacing step 4 of Algorithm 11 with

$$c_6 \leftarrow d - n(\bar{\kappa} c_1 \gg c_5),$$

the multiplication of c_1 by $\bar{\kappa}$ (κ adjusted to $D - N$ bits, shifting by $L - D$ bits to the right), will be done in $O((D-N)^2)$.

Hence, the new work factor decreases to $(D-N)^2 + N(D-N) = (D-N)D$.

5.2.2 Moduli with a Predetermined Portion

RSA [RSA78] moduli with a predetermined portion are used to reduce storage requirements or computations. As mentioned before, such moduli are presently not known to be cryptographically weaker than randomly chosen ones. The first techniques for generating composite moduli were proposed by Vanstone and Zuccherato [VZ95] who presented various ways of specifying $N/4 \leq \ell \leq N/2$ bits of n. Lenstra [Len98] proposed more advanced techniques for specifying up to $N/2$ bits. Based on Lenstra's algorithms, Joye proposed new techniques in [Joy08]. Further works in the area include, for instance, [Kno88, Mei91, Shp06]. We will hereafter recall the method described by Joye (Algorithm 12)which is a small variation of Lenstra's algorithm presented in [Joy08], that perfectly fits our purpose [6].

Joye's Method. The purpose of Lenstra's technique modified by Joye is to obtain an RSA modulus n with a predetermined leading part n_h. Letting $\|n_h\| = H$, we have:

$$n = n_h 2^{N-H} + n_\ell, \text{ for some } 0 < n_\ell < 2^{N-H} \tag{5.1}$$

The algorithm uses the function NextPrime(x) that returns the prime following x (if x is prime then x = NextPrime(x)). Note that because the gap between x and NextPrime(x) is unpredictable, the algorithm may fail to return an n of the form $n = n_h 2^{N-H} + n_\ell$ and will have to be re-launched. We refer the reader to [Len98] for a more formal analysis of this process.

Lemma 5.2 (Bounding n and ω) *Consider the parameters used in Algorithm 12 and let $m = q - \omega$. Then, $n < n_h 2^{N-H} + (1+m)(2^{N-H} - 1)$ and $\omega < 2^{H+1} + 1$.*

Proof: By definition:

$$\omega = \left\lceil \frac{\eta}{p} \right\rceil \Rightarrow \exists\, \alpha < p \text{ such that } \omega = \frac{\eta}{p} + \frac{\alpha}{p}$$

Substituting the value of η, we get:

$$\omega = \frac{n_h 2^{N-H}}{p} + \frac{\alpha}{p} \Rightarrow q = \omega + m = \frac{n_h 2^{N-H}}{p} + \frac{\alpha}{p} + m$$

Thus:

$$n = pq = n_h 2^{N-H} + \alpha + mp < n_h 2^{N-H} + (1+m)p < n_h 2^{N-H} + (1+m)(2^{N-H} - 1)$$

And upper bounding ω we get:

$$\omega < \frac{\eta}{p} + 1 = \frac{n_h 2^{N-H}}{p} + 1 < \frac{n_h 2^{N-H}}{2^{N-H-1}} + 1 = 2n_h + 1 < 2^{H+1} + 1$$

Note that the most significant bit of p must be set to 1, *i.e.* $2^{N-H-1} < p < 2^{N-H} - 1$.

□

It follows directly from Lemma 5.2 that:

$$q = \text{NextPrime}[\omega] \leq \text{NextPrime}[2^{H+1} + 1].$$

6. For the sake of clarity we remove all tests meant to enforce the condition GCD$(e, \phi(n)) = 1$.

Table 5.1: Success rates of Algorithm 12 for $N = 1024$, $H = 512$ and 10^4 experiments.

τ	0	1	2	3	4
$\|\bar{n}_h\|$	503	502	501	500	499
success rate	85.66%	97.96%	99.96%	100%	100%

Algorithm 12: Joye's method

Input: $N, H \leq N/2, n_h < 2^H$

Output: $n = n_h 2^{N-H} + n_\ell$, such that $0 < n_\ell < 2^{N-H}$

1 Generate a random prime p, such that $2^{N-H-1} < p < 2^{N-H} - 1$

2 $\eta \leftarrow n_h 2^{N-H}$

3 $\omega \leftarrow \left\lceil \frac{\eta}{p} \right\rceil$

4 $q \leftarrow \text{NextPrime}(\omega)$

5 $n \leftarrow pq$

6 **return** n

Applying the Prime Number Theorem, we find that $m \simeq \ln(2^{H+1} + 1) \simeq 0.7(H + 1)$. In other words, the $\log_2(m + 1) \simeq \log_2(0.7H + 1.7) < \log_2 H$ least significant bits of n_h are likely to get polluted. We hence rectify the size of n_h to $H - \tau - \log_2 H$ where $\tau \in \mathbb{N}$ is a parameter allowing to reduce the failure probability of Algorithm 12 at the cost of further shortening n_h. For the sake of clarity, we do not integrate these fine-tunings in the description of Algorithm 12 but consider that n_h is composed of a "real" prescribed pattern $\bar{n}_h$ of size $H - \tau - \lceil \log_2 H \rceil$ bits right-padded with $\tau + \lceil \log_2 H \rceil$ zero bits. Various success rates for $N = 1024, H = 512$ are given in Table 5.1. Based on those we recommend to set $\tau = 0$ or $\tau = 1$ and re-launch the generation process if the algorithm fails.

Note. The algorithm's theoretical analysis could be simplified and the failure rate improved if step (4) of Algorithm 12 is replaced by: "If ω is composite then goto 1; else $q \leftarrow \omega$". The quality of the generated primes will also become theoretically uniform because NextPrime favors primes p_i whose distance from the previous prime p_{i-1} is large. This modification will, however, come at the cost of more computation time. The same note is applicable to Algorithm 13 as well.

5.2.3 Barrett-Friendly Moduli

We note that both multiplications in Algorithm 11 are multiplications by n and κ. It is known (*e.g.* [Ber86]) that multiplications by constants can be performed faster than multiplications by arbitrary integers. Our goal is to generate ① a composite n ② whose leading bits do not need to be multiplied and ③ whose associated κ also features a most significant part that does not need to be multiplied. As for the least significant parts of n and κ, these are constants and can hence *independently* benefit of speed-up techniques such as [Ber86]. The algorithm is given for the very common setting $L = D = 2N$. For convenience we introduce a bit-length unit U such that $L = 2N = 4U$.

Example 5.2 *Let* $N = 100$ *and* $L = 200$*:*

```
r   =  1ace38e78e29f              η   =  8000000000001ace38e78e29f
p   =  322a28626f0a7              ω   =  28d356763fe4a
q   =  51a6acec7fcd5              n   =  80000000000a8c93071ac14d9
                                  κ   =  1fffffffffd5cdb3e394fe440
```

Lemma 5.3 *If* $0 < x < 2^{P/2-1}$, *then* $\left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor = 2^{P+1} - 4x$.

Algorithm 13: Barrett-friendly modulus generator

Input: $L = 2N = 4U$

Output: n, an RSA modulus such that $2^{N-1} < n < 2^{N-1} + (0.7U + 2)(2^U - 1)$ whose associated κ is such that $2^{N+1} - 2^{U+1}(1 + 0.7U) < \kappa < 2^{N+1}$

1 Generate a random integer r such that $2^{U-1} < r < 2^U - 1$;
2 $\eta \leftarrow 2^{N-1} + r$
3 Generate a random prime p such that $2^{U-1} < p < 2^U - 1$
4 $\omega \leftarrow \left\lceil \frac{\eta}{p} \right\rceil$
5 $q \leftarrow \text{NextPrime}(\omega)$
6 $n \leftarrow pq$
7 **return** n

Proof: Observe that:

$$\frac{2^{2P}}{2^{P-1} + x} - (2^{P+1} - 4x) = \frac{4x^2}{2^{P-1} + x}. \tag{5.2}$$

Furthermore,

$$\frac{4x^2}{2^{P-1} + x} < 1 \Leftrightarrow 4x^2 - x < 2^{P-1}$$

This is a polynomial of degree 2, that has one positive and one negative root. We assumed $x > 0$, therefore we only need to consider the positive root $x_{\max}$:

$$x_{\max} = \frac{1}{8}\left(1 + \sqrt{1 + 2^{P+4}}\right) > 2^{P/2-1}$$

Therefore, if $x < 2^{P/2-1}$, then the fraction in equation 5.2 is smaller than one. As a consequence, we have

$$\left\lfloor \frac{2^{2P}}{2^{P-1} + x} - (2^{P+1} - 4x) \right\rfloor = \left\lfloor \frac{2^{2P}}{2^{P-1} + x} \right\rfloor - (2^{P+1} - 4x) = 0,$$

as $2^{P+1} - 4x$ is an integer. □

Lemma 5.4 (Bounding n, ω and κ in Algorithm 13) *Consider the parameters used in Algorithm 13 and let $m = q - \omega$. Then, $n < 2^{N-1} + (2 + m)(2^U - 1)$, $2^{N+1} - 2^{U+1}(1 + m) < \kappa < 2^{N+1}$ and $\omega < 2^U + 2$.*

Proof: By definition:

$$\omega = \left\lceil \frac{\eta}{p} \right\rceil, \text{ thus } \exists\, \alpha < p \text{ such that } \omega = \frac{\eta}{p} + \frac{\alpha}{p}.$$

Substituting the value of η, we get:

$$\omega = \frac{2^{N-1} + r}{p} + \frac{\alpha}{p} \Rightarrow q = \omega + m = \frac{2^{N-1}}{p} + \frac{r}{p} + \frac{\alpha}{p} + m.$$

Thus:

$$n = pq = 2^{N-1} + r + \alpha + mp$$

$$\Downarrow$$

$$n < 2^{N-1} + r + (1 + m)p < 2^{N-1} + 2^U - 1 + (1 + m)(2^U - 1) < 2^{N-1} + (2 + m)(2^U - 1).$$

We observe that

$$2^{N-1} + r + \alpha + mp \leq 2^{N-1} + r + mp \Rightarrow \frac{1}{2^{N-1} + r + \alpha + mp} \geq \frac{1}{2^{N-1} + r + mp}.$$

Table 5.2: Success rates of Algorithm 13 for $N = 1024, U = 512$ and 10^4 experiments.

τ	0	1	2	3	4
$\|\bar{n}_h\|$	503	502	501	500	499
success rate	85.16%	97.51%	99.91%	100%	100%

Bounding κ we obtain:

$$\kappa = \left\lfloor \frac{2^L}{n} \right\rfloor > \frac{2^L}{n} - 1 \geq \frac{2^L}{2^{N-1} + r + mp} - 1,$$

Now observe that $r + mp < 2^{N-1}$, therefore we can write

$$\frac{2^L}{2^{N-1} + r + mp} = \frac{2^{2N}}{2^{N-1} + r + mp} = 2^{N+1} \frac{1}{1 + 2^{1-N}(r + mp)} = 2^{N+1} \sum_{\ell=0}^{\infty} (-2)^{\ell(1-N)} (r + mp)^{\ell}$$

This series is convergent, alternating, and the term is strictly decreasing, therefore its sum is bounded below (respectively above) by the partial sum of odd (respectively even) degree S_ℓ. As a consequence,

$$\kappa > S_1 - 1 = 2^{N+1} \left(1 - 2^{1-N}(r + pm)\right) - 1 = 2^{N+1} - 4(r + pm) - 1 > 2^{N+1} - 2^{U+1}(1 + m).$$

We observe that

$$2^{N-1} + r + \alpha + mp > 2^{N-1} \Rightarrow \frac{1}{2^{N-1} + r + \alpha + mp} < \frac{1}{2^{N-1}}.$$

Thus:

$$\kappa \leq \frac{2^L}{n} = \frac{2^L}{2^{N-1} + r + \alpha + mp} < \frac{2^L}{2^{N-1}} < 2^{N+1}.$$

Upper bounding ω we get:

$$\omega < \frac{\eta}{p} + 1 = \frac{2^{N-1} + r}{p} + 1 < \frac{2^{N-1} + 2^{U-1}}{2^{U-1}} + 1 = 2^{N-1-U+1} + 1 + 1 < 2^U + 2.$$

Note that the most significant bit of p must be set to 1, *i.e.* $2^{U-1} < p < 2^U - 1$. □

It follows directly from Lemma 5.4 that:

$$q = \text{NextPrime}[\omega] \leq \text{NextPrime}[2^U + 2] = \text{NextPrime}[2^U + 1].$$

Let n_h denote the predetermined portion of n, *i.e.* $n_h = 2^{U-1}$. Applying the Prime Number Theorem, we obtain $m \simeq \ln(2^U + 1) \simeq 0.7U$. Put differently, the $\log_2(m+2) \simeq \log_2(0.7U+2) < \log_2 U$ least significant bits of n_h are likely to get polluted. We hence rectify the size of n_h to $U - \tau - \log_2 U$ where $\tau \in \mathbb{N}$ is a parameter allowing to reduce the failure probability of Algorithm 13 at the cost of further shortening n_h. For the sake of clarity, we do not integrate these fine-tunings in the description of Algorithm 13 but consider that n_h is composed of a "real" prescribed pattern $\bar{n}_h$ of size $U - \tau - \lceil \log_2 U \rceil$ bits right-padded with $\tau + \lceil \log_2 U \rceil$ zero bits. Various success rates for $N = 1024, U = 512$ are given in Table 5.2. Based on those we recommend to set $\tau = 0$ or $\tau = 1$ and re-launch the generation process if the algorithm fails.

It is easy to see that multiplication by both n and κ is not costly at all. To be more specific, n and κ satisfy the inequalities:

$$2^{N-1} < n < 2^{N-1} + (0.7U + 2)(2^U - 1) \text{ and } 2^{N+1} - 2^{U+1}(1 + 0.7U) < \kappa < 2^{N+1}.$$

As a result, this can double the speed of Barrett reduction[7].

7. A few more complexity bits can be grabbed if the variant described in the note at the end of Section 5.2.2 is used.

Algorithm 14: DSA prime generation

Input: Key lengths P and $Q \leq P$.

Output: Parameters (p, q).

1 Choose a $Q-$bit prime q

2 Choose a $P-$bit prime modulus p such that $p-1$ is a multiple of q

3 **return** (p, q)

Algorithm 15: Barrett-friendly DSA prime generation

Input: Key lengths P and $Q \leq P$.

Output: Parameters (p, q).

1 Generate a $Q-$bit prime as follows:

2 $q \leftarrow \text{NextPrime}(2^{Q-1})$

3 Construct a $P-$bit prime modulus p such that $p-1$ is a multiple of q in the following way:

4 $i \leftarrow 1$

5 $F \leftarrow 2^{P-Q-1}$

6 **while** p is composite **do**

7 $\quad p \leftarrow 2q(F+i)+1$

8 $\quad i++$

9 **end while**

10 **return** (p, q)

5.2.4 Extensions

The parameter generation phase of DL cryptosystems requires the generation of two primes (*e.g.* p and q). Computations modulo these two primes represent important steps within the algorithms. Thus, a modular reduction speedup is necessary. It is thus desirable that both p and q to contain significantly long patterns (*i.e.* many successive 1s or 0s). We will now propose a Barrett-friendly parameter generation approach to do so. For the sake of clarity, we choose a particular algorithm to describe our method: the Digital Signature Algorithm (DSA).

5.2.4.1 Barrett-Friendly DSA Parameters Generation

DSA's parameter generation is presented in Algorithm 14. For the complete description of the DSA, we refer the reader to [fip13].

We suggest a modified DSA prime generation process leveraging the idea of Section 5.2.3. The procedure is described in Algorithm 15.

Lemma 5.5 (Structure of κ_q) *Let κ_q be the κ associated to q. With the notations of Algorithm 15, we have $\kappa_q = 2^{Q+1} - 4\omega$, assuming that $\omega < 2^{\frac{Q}{2}-1}$.*

Proof: Let $z = \frac{p-1}{q}$ and $\omega = q - 2^{Q-1}$. We observe that $\|z\| = P - Q$ and $q = 2^{Q-1} + \omega$. By definition, $\kappa_q = \left\lfloor \frac{2^{L_q}}{q} \right\rfloor$, where $L_q = 2Q$. As we assumed $\omega < 2^{\frac{Q}{2}-1}$, using Lemma 5.3 we have:

$$\kappa_q = \left\lfloor \frac{2^{L_Q}}{q} \right\rfloor = \left\lfloor \frac{2^{2Q}}{2^{Q-1}+\omega} \right\rfloor = 2^{Q+1} - 4\omega.$$

□

The key consequence of Lemma 5.5 is that κ_q consists of a long pattern (a sequence of 1s) concatenated to a short different sequence. The predetermined portion is the complement of $q_h = 2^{Q-\Omega}$, where $\Omega = \|\omega\|$. The computation of κ_q is easy.

Let $L_p = 2P$. By definition, $\kappa_p = \left\lfloor \frac{2^{L_p}}{p} \right\rfloor$.

Lemma 5.6 *Let* $m(n) = \frac{1}{8}\left(n + \sqrt{n^2 + 2^{P+3}n}\right)$. *Let* x *be a positive integer such that* $0 < x < 2^{P-1}$ *and* $m(n) \leq x < m(n+1)$. *Then,*

$$\left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor = 2^{P+1} - 4x + n \quad \textit{and} \quad 0 \leq n < 2^P.$$

Proof: The proof consists of writing the fraction as a geometric series:

$$\begin{aligned}\kappa = \left\lfloor \frac{2^{2P}}{2^{P-1}+x} \right\rfloor &= \left\lfloor 2^{P+1} \sum_{n=0}^{\infty} (-x)^n 2^{n(1-P)} \right\rfloor \\ &= \left\lfloor 2^{P+1}\left(1 - 2^{1-P}x + 2^{2-2P}x^2 - 2^{3-3P}x^3 + \ldots\right)\right\rfloor \\ &= \left\lfloor 2^{P+1} - 4x + 2^{3-P}x^2 - 2^{4-2P}x^3 + \ldots \right\rfloor\end{aligned}$$

Now, $2^{P+1} - 4x$ is always a positive integer, it can therefore be safely taken out of the floor function. None of the remaining terms of the sum is an integer. We have:

$$\kappa = 2^{P+1} - 4x + \left\lfloor \sum_{n=2}^{\infty} (-x)^n 2^{n(1-P)} \right\rfloor.$$

The rightmost term is essentially a sum of shifted versions of powers of x. If x is small, then this contribution quickly vanishes. We now provide an exact value for this sum, by rewriting:

$$\begin{aligned}\kappa &= 2^{P+1} - 4x + \left\lfloor 2^{2-P}x^2 \frac{2^{P-1}}{2^{P-1}+x} \right\rfloor \\ &= 2^{P+1} - 4x + \left\lfloor \frac{4x^2}{2^{P-1}+x} \right\rfloor.\end{aligned}$$

For any positive integer n, we have:

$$\frac{4x^2}{2^{P-1}+x} = n \Leftrightarrow x = \frac{1}{8}\left(n + \sqrt{n^2 + 2^{P+3}n}\right).$$

We assumed $x > 0$, thus we only need to consider the positive root. The leftmost fraction is a strictly increasing function of x as its derivative is > 0. Therefore, the rightmost formula strictly increases with n.
Let $m(n) = \frac{1}{8}\left(n + \sqrt{n^2 + 2^{P+3}n}\right)$ and assume that $m(n) \leq x < m(n+1)$. Then, we have:

$$n \leq \frac{4x^2}{2^{P-1}+x} < n+1$$

Therefore:

$$\left\lfloor \frac{4x^2}{2^{P-1}+x} \right\rfloor = n.$$

Finally, $x < 2^{P-1}$ implies an upper bound on the value of n, which must therefore be smaller than 2^P. □

An illustrative example for $P = 1024$ and $Q = 160$ is given next.

Example 5.3

$$
\begin{aligned}
\omega &= 299 \\
i_p &= 1 \\
L_q &= 2 \cdot 160 \\
q &= 2^{159} + 299 \\
\kappa_q &= 2^{163} - 4 \cdot 299 \\
L_p &= 2 \cdot 1024 = 2^{11} \\
p &= (2^{864} + 2)q + 1 = (2^{864} + 2)(2^{159} + 299) + 1 \\
x &= 2^{60} + 299 \cdot 2^{864} + 2 \cdot 299 + 1 \\
\kappa_p &= 2^{71} \sum_{k=0}^{5} 2^{159k}(-299)^{6-k} - 2^{162} + 2387
\end{aligned}
$$

Thus, multiplication by p, q, κ_p and κ_q is easy.

5.3 Applying Cryptographic Techniques to Error Correction

5.3.1 From Modular Reduction to Polynomial Reduction

Modular reduction (*e.g.* [BGV94, Bri83, Knu81, Mon85]) is the basic building block of many public-key cryptosystems. We refer the reader to [BGV94] for a detailed comparison of various modular reduction strategies.

BCH codes are widely used for error correction in digital systems, memory devices and computer networks. For example, the shortened BCH (48,36,5) was accepted by the U.S. Telecommunications Industry Association as a standard for the cellular Time Division Multiple Access protocol (TDMA) [Shp06]. Another example is BCH(511, 493) which was adopted by International Telecommunication Union as a standard for video conferencing and video phone codecs (Rec. H.26) [Len98]. BCH codes require repeated polynomial reductions modulo the same constant polynomial. This is conceptually very similar to the implementation of public-key cryptography where repeated modular reduction in $\mathbb{Z}_n$ or $\mathbb{Z}_p$ are required for some fixed n or p [Bar87].

It is hence natural to try and transfer the modular reduction expertise developed by cryptographers during the past decades to obtain new BCH speed-up strategies. This work focuses on the "polynomialization" of Barrett's modular reduction algorithm [Bar87]. Barrett's method creates the operation $a \bmod b$ from bit shifts, multiplications and additions in $\mathbb{Z}$. This allows to build modular reduction at very marginal code or silicon costs by leveraging existing hardware or software multipliers.

Reduction modulo fixed multivariate polynomials is also very useful in other fields such as robotics and computer algebra (*e.g.* for computing Gröbner bases).

Organisation: Section 5.3.1.3 presents our main theoretical results, *i.e.* a polynomial variant of [Bar87]. Section 5.3.1.6 recalls the basics of BCH error correcting codes (ECC). Section 5.3.1.6 also describes the integration of the Barrett polynomial variant in a BCH circuit and provides benchmark results.

5.3.1.1 Orders

Definition 5.2 (Monomial Order) *Let P, Q and R be three monomials in ν variables. We say that $\rhd$ is a monomial order if the following conditions are fulfilled:*

- $P \rhd 1$
- $P \rhd Q \Rightarrow \forall R,\ PR \rhd QR$

Example 5.4 *It is straightforward that the lexicographic order on exponent vectors and defined by*

$$\prod_{i=1}^{\nu} x^{a_i} \succ \prod_{i=1}^{\nu} x^{b_i} \Leftrightarrow \exists\, i, a_j = b_j \text{ for } i < j \text{ and } a_i > b_i$$

is a monomial order. We denote it by $\succ$.

5.3.1.2 Terminology

Let $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}} \in \mathbb{Q}[\mathbf{x}]$.

① The leading term of P according to $\rhd$, will be denoted by $\mathrm{lt}(P) = p_0 \prod_{j=1}^{\nu} x_j^{y_{j,0}}$.

② The leading coefficient of P according to $\rhd$ will be denoted by $\mathrm{lc}(P) = p_0 \in \mathbb{Q}$.

③ The quotient $\frac{\mathrm{lt}(P)}{\mathrm{lc}(P)} = \prod_{j=1}^{\nu} x_j^{y_{j,0}}$ is the leading monomial of P according to $\rhd$. We denote it by $\mathrm{lm}(P)$.

The above notations allow to generalize the notion of degree to exponent vectors:

$$\deg(P) = \mathbf{y_0} = \langle y_{0,0}, \ldots, y_{\nu,0} \rangle .$$

Example 5.5 *For $\succ$ and $P(x,y) = 2x_1^2x_2^2 + 11x_1 + 15$, we have that:*

$$lt(P) = 2x_1^2x_2^2,\ lm(P) = x_1^2x_2^2 \text{ and } lc(P) = 2 .$$

5.3.1.3 Barrett's Algorithm for Multivariate Polynomials

We will now adapt Barrett's algorithm to $\mathbb{Q}[\mathbf{x}]$.

Barrett's algorithm and Lemma 5.1 can be generalised to $\mathbb{Q}[\mathbf{x}]$, by shifting polynomials instead of shifting integers.

Definition 5.3 (Polynomial Right Shift) *Let $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}} \in \mathbb{Q}[\mathbf{x}]$ and $\mathbf{a} = \langle a_1, a_2, ..., a_\nu \rangle \in \mathbb{N}^\nu$. We denote*

$$P \gg \mathbf{a} = \sum_{\varphi(\mathbf{a})} p_i \prod_{j=1}^{\nu} x_j^{y_{j,i}-a_i} \in \mathbb{Q}[\mathbf{x}],\ \text{where } \varphi(\mathbf{a}) = \{i,\ \forall j,\ y_{i,j} \geq a_i\} .$$

Example 5.6

$$\text{If } P(x) = 17x^7 + 26x^6 + 37x^4 + 48x^3 + 11,\ \text{then } P \gg \langle 5 \rangle = 17x^2 + 26x .$$

Theorem 5.7 (Barrett's Algorithm for Polynomials) *Let:*

- $P = \sum_{i=0}^{\alpha} p_i \prod_{j=1}^{\nu} x_j^{y_{j,0}} \in \mathbb{Q}[\mathbf{x}]$ *and* $Q = \sum_{i=0}^{\beta} q_i \prod_{j=1}^{\nu} x_j^{w_{j,i}} \in \mathbb{Q}[\mathbf{x}]$ *s.t.* $\mathrm{lm}(Q) \rhd \mathrm{lm}(P)$
- $L \geq \max(w_{i,j}) \in \mathbb{N}, h(L) = \prod_{j=1}^{\nu} x_j^L$ *and* $K = \left\lfloor \frac{h(L)}{P} \right\rfloor$
- $\mathbf{y_0} = \langle y_{1,0}, y_{2,0}, ..., y_{\nu,0} \rangle \in \mathbb{N}^\nu$

Given the above notations, $(K(Q \gg \mathbf{y_0})) \gg (\langle L^\nu \rangle - \mathbf{y_0}) = \left\lfloor \frac{Q}{P} \right\rfloor .$

Proof: Let $G = h(L) \mod P$ and $B = (\mathrm{K}(Q \gg \mathbf{y_0})) = \frac{h(L) - G}{P} \left\lfloor \frac{Q}{\mathrm{lm}(P)} \right\rfloor .$

$$\Downarrow$$

$$B = \frac{\sum_{\varphi(\mathbf{y_0})} q_i \prod_{j=1}^{\nu} x_j^{L+w_{j,i}-y_{j,0}} - G \sum_{\varphi(\mathbf{y_0})} q_i \prod_{j=1}^{\nu} x_j^{w_{j,i}-y_{j,0}}}{P} .$$

Applying the definition of "$\gg$", we obtain

$$B \gg (\langle L \rangle^\nu - \mathbf{y_0}) = \deg_{\geq \mathbf{0}} \frac{Q_{\varphi(\mathbf{y_0})} - G \sum_{\varphi(\mathbf{y_0})} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L}}{P},\ \text{where } \mathbf{0} = \langle 0 \rangle^\nu .$$

Thus,

$$B \gg (\langle L^\nu \rangle - \mathbf{y_0}) = \left\lfloor \frac{Q_{\varphi(\mathbf{y_0})}}{P} \right\rfloor - \deg_{\geq \mathbf{0}} \frac{G}{P} \sum_{\varphi(\mathbf{y_0})} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} = \left\lfloor \frac{Q_{\varphi(\mathbf{y_0})}}{P} \right\rfloor .$$

We know that

$$P \rhd G \text{ and } L \geq \max(w_{i,j}),\ \text{therefore } \deg_{\geq \mathbf{0}} \frac{G}{P} \sum_{\varphi(\mathbf{y_0})} q_i \prod_{j=1}^{\nu} x^{w_{j,i}-L} = 0 .$$

Let $\bar{Q}$ be the irreducible polynomial with respect to P, obtained by removing from Q the terms that exceed $\mathrm{lm}(P)$.

$$\left\lfloor \frac{Q_{\varphi(\mathbf{y})}}{P} \right\rfloor = \frac{Q_{\varphi(\mathbf{y})} - (Q_{\varphi(\mathbf{y})} \bmod P)}{P} = \frac{(Q-\bar{Q})((Q-\bar{Q}) \bmod P)}{P}.$$

Hence,

$$B \gg (\langle L \rangle^{\nu} - \mathbf{y_0}) = \frac{(Q-\bar{Q})((Q-\bar{Q}) \bmod P)}{P}$$

$$\Downarrow$$

$$B \gg (\langle L \rangle^{\nu} - \mathbf{y_0}) = \left\lfloor \frac{Q}{P} \right\rfloor - \frac{\bar{Q} - \bar{Q} \bmod P}{P} = \left\lfloor \frac{Q}{P} \right\rfloor.$$

□

Algorithm 16: Polynomial Barrett Algorithm

Input: $P, Q \in \mathbb{Q}[\mathbf{x}]$ s.t. $P \rhd Q$

$h(L) = \mathbf{x}^L, \mathbf{y_0} = \deg P$ and $K = h(L) \bmod P$, where $\deg Q \leq \langle L, \ldots, L \rangle$

Output: $R = Q \bmod P$

1 $B \leftarrow (K(Q \gg \mathbf{y_0})) \gg (L - \mathbf{y_0})$

2 $R \leftarrow Q - BP$

3 **return** R

Remark. Let $Q = \sum_{i=0}^{\alpha} q_{i,j} \prod_{j=1}^{\nu} x_j^{w_{j,i}}$, $K = \sum_{i=0}^{\beta} k_{i,j} \prod_{j=1}^{\nu} x_j^{t_{j,i}}$, $\mathbf{y} = \langle y_1, ..., y_\nu \rangle$ and $\mathbf{z} = \langle z_1, ..., z_\nu \rangle$.

Let us have a closer look at the expression $B = (K(Q \gg \mathbf{y})) \gg \mathbf{z}$.

Given the final shifting by $\mathbf{z}$, the multiplication of K by $Q \gg \mathbf{y}$ can be optimised by being only partially accomplished. Indeed, during multiplication, we only have to form monomials whose exponent vectors $\mathbf{b} = \mathbf{w_i} + \mathbf{t_{i'}} - \mathbf{y} - \mathbf{z} = \langle b_1, ..., b_\nu \rangle$ are such that $b_j \geq 0$ for $1 \leq j \leq \nu$.

We implicitly apply the above in the following example.

Example 5.7 *Let*

$$\begin{aligned} \rhd &= \succ \\ P &= x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1 \\ Q &= x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3. \end{aligned}$$

We let $L = 6$ and we observe that $\nu = 2$. We pre-compute K:

$$\begin{aligned} K = {} & x_1^4x_2^4 - x_1^4x_2^2 + x_1^4 - 2x_1^3x_2^4 - 2x_1^3x_2^3 + 3x_1^3x_2^2 + 4x_1^3x_2 - 4x_1^3 + \\ & 4x_1^2x_2^4 + 8x_1^2x_2^3 - 5x_1^2x_2^2 - 20x_1^2x_2 + 3x_1^2 - 8x_1x_2^4 - 24x_1x_2^3 + \\ & 68x_1x_2 + 36x_1 + 16x_2^4 + 64x_2^3 + 36x_2^2 - 184x_2 - 239. \end{aligned}$$

We first shift Q by $\mathbf{y_0} = \langle 2, 2 \rangle$, which is the vector of exponents for $lm(P)$.

$$Q \gg \mathbf{y_0} = (x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3) \gg \langle 2, 2 \rangle = (x_1x_2 + 1).$$

Then, we compute $K(x_1x_2 + 1) = x_1^5x_2^5 - 2x_1^4x_2^5 - x^4y^4 + \{terms \prec x_1^4x_2^4\}$.

This result shifted by $\langle L\rangle^{\nu} - \mathbf{y_0} = \langle 6,6\rangle - \langle 2,2\rangle = \langle 4,4\rangle$ *to the right gives:*

$$A = x_1^5x_2^5 - 2x_1^4x_2^5 - x^4y^4 + \{terms \succ x_1^4x_2^4\} \gg \langle 4,4\rangle = x_1x_2 - 2x_2 - 1\,.$$

It is easy to verify that:

$$Q - PA =$$
$$= (x_1^3x_2^3 - 2x_1^3 + x_1^2x_2^2 + 3) - (x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1)(x_1x_2 - 2x_2 - 1)$$
$$\Downarrow$$
$$Q - PA = 4x_1x_2^3 + 6x_1x_2^2 - x_1^3x_2 + x_1^2x_2 + 3x_1x_2 + 2x_2 - 2x_1^3 + x_1^2 + x_1 + 4 \prec P\,.$$

5.3.1.4 Polynomial Barrett Complexity.

We decompose the algorithm's analysis into steps and determine at each step the *cost* and the *size* of the result. Size is measured in the number of terms. In all the following we assume that polynomial multiplication is performed using traditional cross product. Faster (*e.g.* ν-dimensional FFT [TAL93]) polynomial multiplication strategies may grandly improve the following complexities for asymptotically increasing $\mathbf{L}$ and ν.

Given our focus on on-line operations we do not count the effort required to compute K (that we assume given). We also do not account for the partial multiplication trick for the sake of clarity and conciseness.

Let $\omega \in \mathbb{Z}^{\nu}$, in this appendix we denote by $||\omega||$ the quantity

$$||\omega|| = \prod_{j=1}^{\nu} \omega_j \in \mathbb{Z}\,.$$

1. $Q \gg \mathbf{y_0}$.
   (a) **Cost**: $\mathrm{lm}(Q)$ is at most $\langle L, ..., L\rangle$ hence Q has at most L^{ν} monomials. Shifting discards all monomials having exponent vectors ω for which $\exists\, j$ such that $\omega_j < y_{j,0}$. The number of such discarded monomials is $\mathrm{O}(||\mathbf{y_0}||)$, hence the overall complexity of this step is:
   $$\mathrm{cost}_1 = \mathrm{O}((L^{\nu} - ||\mathbf{y_0}||)\nu) = \mathrm{O}((L^{\nu} - \prod_{j=1}^{\nu} y_{j,0})\nu)\,.$$
   (b) **Size**: The number of monomials remaining after the shift is
   $$\mathrm{size}_1 = \mathrm{O}(L^{\nu} - ||\mathbf{y_0}||) = \mathrm{O}(L^{\nu} - \prod_{j=1}^{\nu} y_{j,0})\,.$$
2. $K(Q \gg \mathbf{y_0})$.
   Because K is the result of the division of $h(L) = \prod_{j=1}^{\nu} x_j^L$ by P, the leading term of K has an exponent vector equal to $\mathbf{L} - \mathbf{y_0}$. This means that K's second biggest term can be $x_1^{L-y_{1,0}} \prod_{j=2}^{\nu} x_j^L$. Hence, the size of K is
   $$\mathrm{size}_K = \mathrm{O}((L - y_{1,0})L^{\nu-1})\,.$$
   (a) **Cost**: The cost of computing $K(Q \gg \mathbf{y_0})$ is
   $$\mathrm{cost}_2 = \mathrm{O}(\nu \times \mathrm{size}_1 \times \mathrm{size}_K)\,.$$
   (b) **Size**: The size of $K(Q \gg \mathbf{y_0})$ is determined by $\mathrm{lm}(K(Q \gg \mathbf{y_0})) = \mathrm{lm}(K) \times \mathrm{lm}(Q \gg \mathbf{y_0})$ which has the exponent vector $\mathbf{u} = (\mathbf{L} - \mathbf{y_0}) + \langle L - y_{1,0}, L, ..., L\rangle$.
   $$\mathrm{size}_2 = \mathrm{O}(||\mathbf{u}||) = \mathrm{O}(2(L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0}))$$
   $$= \mathrm{O}((L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0})).$$

3. $B = (K(Q \gg \mathbf{y_0})) \gg (\mathbf{L} - \mathbf{y_0})$

   (a) **Cost**: The number of discarded monomials is $\mathrm{O}(||\mathbf{L} - \mathbf{y_0}||)$, hence the cost of this step is

   $$\mathrm{cost}_3 = \mathrm{O}((2(L - y_{1,0}) \prod_{j=2}^{\nu} (2L - y_{j,0}) - \prod_{j=1}^{\nu} (L - y_{j,0}))\nu)\,.$$

   (b) **Size**: The leading monomial of B has the exponent vector $\mathbf{u} - \mathbf{L} - \mathbf{y_0}$ which is equal to $\langle L - y_{1,0}, L, ..., L\rangle$. We thus have $\mathrm{size}_B = \mathrm{size}_K$.

4. BP
   The cost of this step is

   $$\mathrm{cost}_4 = \mathrm{O}(\nu \times \mathrm{size}_B \times \mathrm{size}_P) = \mathrm{O}(\nu \times \mathrm{size}_B \times ||\mathbf{y_0}||)\,.$$

5. Final subtraction $Q - BP$
   The cost of polynomial subtraction is negligible with respect to cost_4.
6. **Overall complexity**
   The algorithm's overall complexity is hence

   $$\max(\mathrm{cost}_1, \mathrm{cost}_2, \mathrm{cost}_3, \mathrm{cost}_4) = \mathrm{cost}_2\,.$$

5.3.1.5 Dynamic Constant Scaling in $\mathbb{C}[\mathbf{X}]$

Lemma 5.8 *If* $0 \leq u \leq L$, *then* $\bar{K} = K \gg \langle u\rangle^{\nu} = \left\lfloor \frac{h(L-u)}{P} \right\rfloor$.

Proof: $K = \left\lfloor \frac{h(L)}{P} \right\rfloor \Rightarrow\ K = \frac{h(L) - h(L) \bmod P}{P}$.

Let $G = h(L) \bmod P \Rightarrow K = \dfrac{\prod_{j=1}^{\nu} {x_j}^L - G}{P}$.

Since

$$\langle u\rangle^{\nu} \in \mathbb{N}^{\nu} \Rightarrow K \gg \langle u\rangle^{\nu} = \deg_{\geq \mathbf{0}} \frac{\prod_{j=1}^{\nu} {x_j}^{L-u} - G_{\varphi(\langle u\rangle^{\nu})}}{P}$$

$$\Downarrow$$

$$K \gg \langle u\rangle^{\nu} = \deg_{\geq \mathbf{0}} \frac{\prod_{j=1}^{\nu} {x_j}^{L-u}}{P} - \deg_{\geq \mathbf{0}} \frac{G_{\varphi(\langle u\rangle^{\nu})}}{P}\,.$$

We know that $P \rhd G$, thus $P \rhd G_{\varphi(\langle u\rangle^{\nu})}$, thus $\deg_{\geq \mathbf{0}} \dfrac{G_{\varphi(\langle u\rangle^{\nu})}}{P} = 0$.

Finally,

$$K \gg \langle u\rangle^{\nu} = \left\lfloor \frac{\prod_{j=1}^{\nu} {x_j}^{L-u}}{P} \right\rfloor = \left\lfloor \frac{h(L-u)}{P} \right\rfloor\,.$$

□

Example 5.8 *Let*

$$\begin{aligned} \rhd &= \succ \\ P &= x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1 \\ Q &= x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3. \end{aligned}$$

We let $u = 4$ and we observe that $\nu = 2$. We pre-compute $\bar{K}$:

$$\bar{K} = x_1^2x_2^2 - x_1^2 - 2x_1x_2^2 - 2x_1x_2 + 3x_1 + 4x_2^2 + 8x_2 - 5.$$

We first shift Q by $\mathbf{y_0} = \langle 2, 2\rangle$, which is the vector of exponents for $lm(P)$.

$$Q \gg \mathbf{y_0} = (x_1^3x_2^3 - 2x_1^3 + x_2^2x_2^2 + 3) \gg \langle 2, 2\rangle = (x_1x_2 + 1)\,.$$

Then, we compute $\bar{K}(x_1x_2 + 1) = x_1^3x_2^3 - 2x_1^2x_2^3 - x_1^2x_2^2 + \{terms \prec x_1^2x_2^2\}$.
This result shifted by $\langle u\rangle^\nu - \mathbf{y_0} = \langle 4, 4\rangle - \langle 2, 2\rangle = \langle 2, 2\rangle$ to the right gives:

$$A = x_1^3x_2^3 - 2x_1^2x_2^3 - x_1^2x_2^2 + \{terms \succ x_1^2x_2^2\} \gg \langle 2, 2\rangle = x_1x_2 - 2x_2 - 1\,.$$

It is easy to verify that:

$$Q - PA =$$
$$= (x_1^3x_2^3 - 2x_1^3 + x_1^2x_2^2 + 3) - (x_1^2x_2^2 + x_1^2 + 2x_1x_2^2 + 2x_1x_2 + x_1 + 1)(x_1x_2 - 2x_2 - 1)$$
$$\Downarrow$$
$$Q - PA = 4x_1x_2^3 + 6x_1x_2^2 - x_1^3x_2 + x_1^2x_2 + 3x_1x_2 + 2x_2 - 2x_1^3 + x_1^2 + x_1 + 4 \prec P\,.$$

5.3.1.6 Application to BCH Codes

General Remarks.
BCH codes are cyclic codes that form a large class of multiple random error-correcting codes. Originally discovered as binary codes of length $2^m - 1$, BCH codes were subsequently extended to non-binary settings. Binary BCH codes are a generalization of Hamming codes, discovered by Hocquenghem, Bose and Chaudhuri [Ber86, Bri83] featuring a better error correction capability. Gorestein and Zierler [Joy08] generalised BCH codes to p^m symbols, for p prime. Two important BCH code sub-classes exist. Typical representatives of these sub-classes are Hamming codes (binary BCH) and Reed Solomon codes (non-binary BCH).

Terminology.
We further refer to the vectors of an error correction code as *codewords*. The codewords' size is called the *length* of the code. The *distance* between two codewords is the number of coordinates at which they differ. The *minimum distance* of a code is the minimum distance between two codewords.

Recall that a *primitive element* of a finite field is a generator of the multiplicative group of the field.

5.3.1.6.1 BCH Preliminaries.

Definition 5.4 *Let $m \geq 3$. For a length $n = 2^m - 1$, a distance d and a primitive element $\alpha \in \mathbb{F}_{2^m}^*$, we define the binary BCH code:*

$$\mathrm{BCH}(n, d) = \{(c_0, c_1, ..., c_{n-1}) \in \mathbb{F}_2^n \mid c(x) = \sum_{i=0}^{n-1} c_i x^i \text{ satisfies}$$

$$c(\alpha) = c(\alpha^2) = ... = c(\alpha^{d-1}) = 0\}\,.$$

Let $m \geq 3$ and $0 < t < 2^{m-1}$ be two integers. There exists a binary BCH code (called a $t-$error correcting BCH code) with parameters $n = 2^m - 1$ (the block length), $n - k \leq mt$ (the number of parity-check digits) and $d \geq 2t + 1$ (the minimum distance).

Definition 5.5 *Let α be a primitive element in $\mathbb{F}_{2^m}$. The* generator polynomial $g(x) \in \mathbb{F}_2[x]$ *of the $t-$error-correcting BCH code of length 2^{m-1} is the lowest-degree polynomial in $\mathbb{F}_2[x]$ having roots $\alpha, \alpha^2, ..., \alpha^{2t}$.*

The degree of $g(x)$, which is the number of parity-check digits $n - k$, is at most mt.

Let $i \in \mathbb{N}$ and denote $i = 2^r j$ for odd j and $r \geq 1$. Then $\alpha^i = (\alpha^j)^{2^r}$ is a conjugate of α^j which implies that α^i and α^j have the same minimal polynomial, and therefore $\phi_i(x) = \phi_j(x)$. Consequently, the generator polynomial $g(x)$ of the t-error correcting BCH code can be written as follow:

$$g(x) = \text{lcm}\{\phi_1(x), \phi_3(x), \phi_5(x), ..., \phi_{2t-1}(x)\}\,.$$

Definition 5.6 (Codeword) *An $n-$tuple $c = (c_0, c_1, ..., c_{n-1}) \in \mathbb{F}_{2^n}$ is a codeword if the polynomial $c(x) = \sum c_i x^i$ has $\alpha, \alpha^2, ..., \alpha^{2t}$ as its roots.*

Definition 5.7 (Dual Code) *Given a linear code $C \subset \mathbb{F}_q^n$ of length n, the* dual code *of C (denoted by $C^\perp$) is defined to be the set of those vectors in $\mathbb{F}_q^n$ which are orthogonal [8] to every codeword of C, i.e.:*

$$C^\perp = \{v \in \mathbb{F}_q^n | v \cdot c = 0, \forall c \in C\}\,.$$

As α^i is a root of $c(x)$ for $1 \leq i \leq 2t$, then $c(\alpha^i) = \sum c_i \alpha^{ij}$. This equality can be written as a matrix product and results in the next property:

Property: If $c = (c_0, c_1, ..., c_{n-1})$ is a codeword, then the parity-check matrix H of this code satisfies $c \cdot H^T = 0$, where:

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \dots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \dots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \dots & (\alpha^{2t})^{n-1} \end{pmatrix}.$$

If $c \cdot H^T = 0$, then $c(\alpha^i) = 0$.

Remark A parity check matrix of a linear block code is a generator matrix of the dual code. Therefore, c must be a codeword of the $t-$error correcting BCH code. If each entry of H is replaced by its corresponding $m-$tuple over $\mathbb{F}_2$ arranged in column form, we obtain a binary *parity-check matrix* for the code.

Definition 5.8 (Systematic Encoding) *In systematic encoding, information and check bits are concatenated to form the message transmitted over the noisy channel.*

The speed-up we described applies to systematic BCH coding only.

Consider an (n, k) BCH code. Let $m(x)$ be the information polynomial to be coded and $m'x^{n-k} = m(x)$.

We can write $m'(x)$ as $m(x)g(x) + b(x)$.

The message $m(x)$ is coded as $c(x) = m'(x) - b(x)$ [9].

BCH Decoding. *Syndrome decoding* is a decoding process for linear codes using the parity-check matrix.

Definition 5.9 (Syndrome) *Let c be the emitted word and r the received one. We call the quantity $S(r) = r \cdot H^T$ the syndrome of r .*

If $r \cdot H^T = 0$ then no errors occurred, with overwhelming probability. If $r \cdot H^T \neq 0$, at least one error occurred and $r = c + e$, where e is an error vector. Note that $S(r) = S(e)$. The syndrome circuit consists of $2t$ components in $\mathbb{F}_{2^m}$. To correct t errors, the syndrome has to be a $2t$-tuple of the form $S = (S_1, S_2, \cdots, S_{2t})$.

8. The scalar product of the two vectors is equal to 0.
9. where $b(x)$ is the remainder of the division of $c(x)$ by $g(x)$

Syndrome. In the polynomial setting, S_i is obtained by evaluating r at the roots of $g(x)$.

Indeed, letting $r(x) = c(x) + e(x)$, we have

$$S_i = r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) = \sum_{k=0}^{\nu-1} e_k \alpha^{ik}, \text{ for } i \leq 1 \leq 2t\,.$$

Suppose that r has ν errors denoted e_{j_i}. Then

$$S_i = \sum_{j=1}^{\nu} e_{j_i} (\alpha^i)^{j_\ell} = \sum_{j=1}^{\nu} e_{j_i} (\alpha^{j_\ell})^i\,.$$

Error Location. Let $X_\ell = \alpha^{j_\ell}$. Then, for binary BCH codes, we have $S_i = \sum_{j=1}^{\nu} X_\ell^i$. The X_ℓ's are called *error locators* and the *error locator polynomial* is defined as:

$$\Lambda(x) = \prod_{\ell=1}^{\nu} (1 - X_\ell) = 1 + \Lambda_1 x + ... + \Lambda_\nu x^\nu\,.$$

Note that the roots of $\Lambda(x)$ point out errors' places and the number of errors ν is unknown.

There are several ways to compute $\Lambda(x)$, *e.g.* Peterson's algorithm [Kno88] or Berlekamp-Massey algorithm [Knu81]. Chien's search method [Mei91] is applied to determine the roots of $\Lambda(x)$.

Peterson's Algorithm. Peterson's Algorithm (Algorithm 17) solves a set of linear equations to find the value of the coefficients $\sigma_1, \sigma_2, \ldots \sigma_t$.

$$\Lambda(x) = \prod_{\ell=1}^{\nu} (1 + \alpha^{jl}) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_t x^t$$

Algorithm 17: Peterson's Algorithm

1 Initialization $\nu \leftarrow t$

2 Compute the determinant of S

$$\det(S) \leftarrow \det \begin{pmatrix} S_1 & S_2 & \cdots & S_t \\ S_2 & S_3 & \cdots & S_{t+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_t & S_{t+1} & \cdots & S_{2t-1} \end{pmatrix}$$

3 Find the correct value of ν

- $\det(S) \neq 0 \longrightarrow$ go to step 4
- $\det(S) = 0 \longrightarrow$
  - **if** $\nu = 0$ **then**
    - The error locator polynomial is empty
    - stop
  - **else**
    - $\nu \longleftarrow \nu - 1$, and then repeat step 2
  - **end if**

4 Invert S and compute $\Lambda(x)$

$$\begin{bmatrix} \sigma_\nu \\ \sigma_{\nu-1} \\ \vdots \\ \sigma_1 \end{bmatrix} = S^{-1} \times \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu} \end{bmatrix}$$

At the beginning of Algorithm 17, the number of errors is undefined. Hence the maximum number of errors to resolve the linear equations generated by the matrix S is assumed. Let this number be $i = \nu = t$.

Chien's Error Search. Chien search finds the roots of $\Lambda(x)$ by brute force [Bri83, Mei91]. The algorithm evaluates $\Lambda(\alpha^i)$ for $i = 1, 2, \ldots, 2^m - 1$. Whenever the result is zero, the algorithm assumes that an error occurred, thus the position of that error is located. A way to reduce the complexity of Chien search circuits stems from Equation 5.3 for $\Lambda(\alpha^{i+1})$.

$$\begin{aligned}
\Lambda(\alpha^i) &= 1 + \sigma_1\ \alpha^i + \sigma_2\ (\alpha^i)^2 + \cdots + \sigma_t\ (\alpha^i)^t \\
&= 1 + \sigma_1\ \alpha^i + \sigma_2\ \alpha^{2i} + \cdots + \sigma_t\ \alpha^{it} \\
\Lambda(\alpha^{i+1}) &= 1 + \sigma_1\ \alpha^{i+1} + \sigma_2\ (\alpha^{i+1})^2 + \cdots + \sigma_t\ (\alpha^{i+1})^t \\
&= 1 + \alpha\ (\sigma_1\ \alpha^i) + \alpha^2\ (\sigma_2\ \alpha^{2i}) + \cdots + \alpha^t\ (\sigma_t\ \alpha^{it})
\end{aligned} \tag{5.3}$$

5.3.1.6.2 Implementation and Results. To evaluate the efficiency of Barrett's modular division in hardware, the BCH$(15, 7, 2)$ was chosen as a case study code. Four BCH encoder versions were designed and synthesized. Results are presented in detail in the coming sections.

Standard Architecture. The `BCH-Standard` architecture consists of applying the modular division using shifts and XORs. Initially, to determine the degree of the input polynomials, each bit [10] of the dividend and of the divisor are checked until the first bit one is found. Then, the two polynomials are left-aligned (*i.e.*, the two most significant ones are aligned) and XORed. The resulting polynomial is right shifted and again left-aligned with the dividend and XORed. This process is repeated until the dividend and the resulting polynomial are right-aligned. The final resulting polynomial represents the remainder of the division. Algorithm 18 provides the pseudocode for the standard architecture.

Algorithm 18: Standard modular division (`BCH-Standard`)

Input: P, Q
Output: `remainder` $= Q \bmod P$

```
1  diff_degree ← deg(Q) − deg(P)
2  shift_counter ← diff_degree + 1
3  shift_divisor ← P ≪ diff_degree
4  remainder ← Q
5  while shift_counter ≠ 0 do
6  |  if remainder[p_degree + shift_counter − 1] = 1 then
7  |  |  shift_counter ← shift_counter − 1  shift_divisor ← shift_divisor ≫ 1
8  |  end if
9  end while
10 return remainder
```

LFSR and Improved LFSR Architectures. The `BCH-LFSR` design is composed from a control unit and a Linear-Feedback Shift Register (LFSR) submodule. The LFSR submodule receives the input data serially and shifts it to the internal registers, controlled by the enable signal. The LFSR's size (the number of parallel flip-flops) is defined by the BCH parameters n and k, *i.e.*, size(LFSR) = $n - k$, and the LFSR registers are called d_i, enumerated from 0 to $n - k - 1$. The feedback value is defined by the XOR of the last LFSR register (d_{nk-1}) and the input data. The feedback connections are defined by the generator polynomial $g(x)$. In the case of BCH$(15, 7, 2)$, $g(x) = x^8 + x^7 + x^6 + x^4 + 1$, therefore the input of registers d_0, d_4, d_6 and d_7 are XORed with the feedback value. As shown in Fig. 5.6, the multiplexer that selects the bits to compose the final codeword is controlled by the counter. The LFSR is shifted k times with the feedback connections enabled. After that, the LFSR state contains the result of the modular division, therefore the bits can be serially shifted out from the LFSR register.

To calculate the correct codeword, the LFSR must shift the input data during k clock cycles. After that, the output is serially composed by $n - k$ extra shifts. This means that the LFSR implementation's total

10. Considered in big endian order.

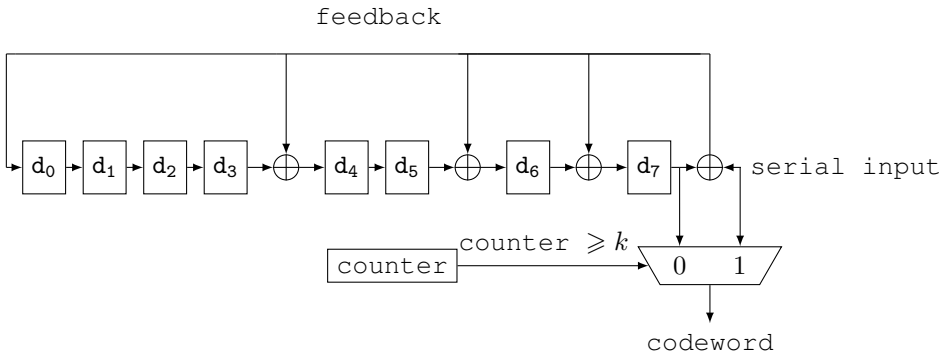


Figure 5.6: Standard LFSR architecture block diagram. (Design `BCH-LFSR`)

latency is n clock cycles. Nevertheless, it is possible to save $n - k - 1$ clock cycles by outputting the LFSR in parallel from the sub-module to the control unit after k iterations, while during the k first cycles the input data is shifted to the output register, as we perform systematic BCH encoding. This decreases the total latency to $k + 1$ clock cycles. This method was applied to the `BCH-LFSR-improved` design depicted in Fig. 5.7.

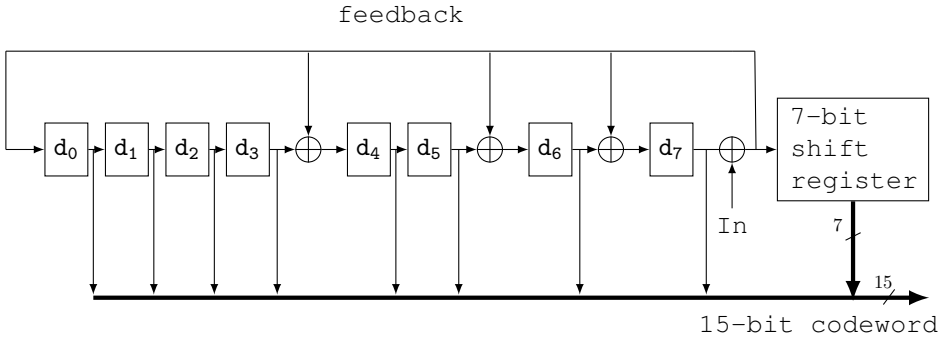


Figure 5.7: Improved LFSR architecture block diagram. `In` denotes the module's serial input. (Design `BCH-LFSR-improved`)

Barrett Architecture. The LFSR submodule can be replaced by the Barrett submodule to evaluate its performance. The idea is that Barrett operations can be broken down into up to $k + 1$ pipeline stages, to match the LFSR's latency. The fact that Barrett operations can be easily pipelined drastically increases the final throughput, while both LFSR implementations do not allow for pipelining.

In the Barrett submodule, the constants y_0, L, and K are pre-computed and are defined as parameters of the block. Since the Barrett parameter P is defined as the generator polynomial, P does not need to be defined as an input, which saves registers. As previously stated, Barrett operations were cut down to k iterations (in our example, $k = 7$). The first register in the pipeline stores the result of $Q \gg y_0$. The multiplication by K is the most costly operation, taking 5 clock cycles to complete. Each cycle operates on 3 bits, shifting and XORing at each one bit of K, according to the rules of multiplication. The last operation simply computes the intermediate result from the multiplication left-shifted by $L - y_0$.

Performance. As mentioned previously in this thesis, the gate equivalent (GE) metric was calculated by dividing the total cell area of each design by the size of the smallest NAND-2 of the digital library. This metric allows comparing area figures without the impact of the technology node size. `BCH-Barrett` presented comparable area with the smallest design, the `BCH-LFSR`. Although the `BCH-Barrett` does not reach the maximum clock frequency, it can be seen from Table 5.7 that it actually reaches the best throughput, around 2.3Gbps. This is mainly achieved by Barrett parallelizable operations, allowing the design to be easily pipelined. Moreover, Barrett consumes the less power among the four designs.

Table 5.3: Synthesis results of the four BCH encoder designs.

Design	Gate Instances	Gate Equivalent	Max Frequency (MHz)	Throughput (Mbps)	Power (nW)
BCH-Standard	310	447	741	690	978
BCH-LFSR	155	223	1043	972	920
BCH-LFSR-improved	160	236	1043	2080	952
BCH-Barrett	194	260	655	9150	512
BCH-Barrett-pipelined	426	591	995	13900	2208

5.3.2 A Number-Theoretic Error-Correcting Code

Error-correcting codes (ECCs) are essential to ensure reliable communication. ECCs work by adding redundancy which enables detecting and correcting mistakes in received data. This extra information is, of course, costly and it is important to keep it to a minimum: there is a trade-off between how much data is added for error correction purposes (bandwidth), and the number of errors that can be corrected (correction capacity).

Shannon showed [Sha48] in 1948 that it is in theory possible to encode messages with a minimal number of extra bits [11]. Two years later, Hamming [Ham50] proposed a construction inspired by parity codes, which provided both error detection and error correction. Subsequent research saw the emergence of more efficient codes, such as Reed-Muller [Mul54, Ree54] and Reed-Solomon [RS60]. The latest were generalized by Goppa [Gop81]. These codes are known as algebraic-geometric codes.

Convolutional codes were first presented in 1955 [Eli55], while recursive systematic convolutional codes [BGT93] were introduced in 1991. Turbo codes [BGT93] were indeed revolutionary, given their closeness to the channel capacity ("near Shannon limit").

Organisation: We further present a new error-correcting code, as well as a form of message size improvement based on the hybrid use of two ECCs one of which is inspired by the Naccache-Stern (NS) cryptosystem [NS97, CMNS08]. For some codes and parameter choices, the resulting hybrid codes outperform the two underlying ECCs.

The proposed ECC is unusual because it is based on number theory rather than on binary operations.

5.3.2.1 Preliminaries

5.3.2.1.1 Notations. Let $\mathfrak{P} = \{p_1 = 2, \dots\}$ be the ordered set of prime numbers. Let $\gamma \geq 2$ be an encoding base. For any $m \in \mathbb{N}$ (the "message"), let $\{m_i\}$ be the digits of m in base γ *i.e.*:

$$m = \sum_{i=0}^{k-1} \gamma^i m_i \qquad m_i \in [0, \gamma - 1], \quad k = \lceil \log_\gamma m \rceil$$

We denote by $h(x)$ the Hamming weight of x, *i.e.* the sum of x's digits in base 2, and, by $\|y\|$ the bit-length of y.

5.3.2.1.2 Error-Correcting Codes. Let $\mathcal{M} = \{0,1\}^k$ be the set of messages, $\mathcal{C} = \{0,1\}^n$ the set of encoded messages. Let $\mathcal{P}$ be a parameter set.

Definition 5.10 (Error-Correcting Code) *An* error-correcting code *is a couple of algorithms:*
- *An algorithm μ, taking as input some message $m \in \mathcal{M}$, as well as some public parameters* $\mathsf{params} \in \mathcal{P}$*, and outputting $c \in \mathcal{C}$.*
- *An algorithm μ^{-1}, taking as input $\tilde{c} \in \mathcal{C}$ as well as parameters* $\mathsf{params} \in \mathcal{P}$*, and outputting $m \in \mathcal{M} \cup \{\perp\}$. The $\perp$ symbol indicates that decoding failed.*

Definition 5.11 (Correction Capacity) *Let $(\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$ be an error-correcting code. There exists an integer $t \geq 0$ and some parameters* $\mathsf{params} \in \mathcal{P}$ *such that, for all $e \in \{0,1\}^n$ such that $h(e) \leq t$,*

$$\mu^{-1}\left(\mu\left(m, \mathsf{params}\right) \oplus e, \mathsf{params}\right) = m, \qquad \forall m \in \mathcal{M}$$

and for all e such that $h(e) > t$,

$$\mu^{-1}\left(\mu\left(m, \mathsf{params}\right) \oplus e, \mathsf{params}\right) \neq m, \qquad \forall m \in \mathcal{M}.$$

t is called the correction capacity *of $(\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$.*

Definition 5.12 *A code of message length k, of codeword length n and with a correction capacity t is called an (n, k, t)-code. The ratio $\rho = \frac{n}{k}$ is called the code's* expansion rate.

11. Shannon's theorem states that the best achievable expansion rate is $1 - H_2(p_b)$, where H_2 is binary entropy and p_b is the acceptable error rate.

5.3.2.2 A New Error-Correcting Code

Consider in this section an existing (n, k, t)-code $C = (\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$. For instance C can be a Reed-Muller code. We describe how the new (n', k, t)-code $C' = (\nu, \nu^{-1}, \mathcal{M}, \mathcal{C}', \mathcal{P}')$ is constructed.

Parameter Generation: To correct t errors in a k-bit message, we generate a prime p such that:

$$2 \cdot p_k^{2t} < p < 4 \cdot p_k^{2t} \tag{5.4}$$

As we will later see, the size of p is obtained by bounding the worst case in which all errors affect the end of the message. p is a part of $\mathcal{P}'$.

Encoding: Assume we wish to transmit a k-bit message m over a noisy channel. Let $\gamma = 2$ so that m_i denote the i-th bit of m, and define:

$$c(m) := \prod_{i=1}^{k} p_i^{m_i} \bmod p\,. \tag{5.5}$$

The integer generated by Equation 5.5 is encoded using C to yield $\mu(c(m))$. Finally, the encoded message $\nu(m)$ transmitted over the noisy channel is defined as:

$$\mu(m) := m \| \mu(c(m))\,. \tag{5.6}$$

Note that, if we were to use C directly, we would have encoded m (and not c). The value c is, in most practical situations, much shorter than m. As is explained in Section 5.3.2.2.1, c is smaller than m (except the cases in which m is very small and which are not interesting in practice) and thereby requires fewer extra bits for correction. For appropriate parameter choices, this provides a more efficient encoding, as compared to C.

Decoding: Let α be the received [12] message. Assume that at most t errors occurred during transmission:

$$\alpha = \nu(m) \oplus e = m' \| (\mu(c(m)) \oplus e')$$

where the error vector e is such that $h(e) = h(m' \oplus m) + h(e') \leq t$.

Since $c(m)$ is encoded with a t-error-capacity code, we can recover the correct value of $c(m)$ from $\mu(c(m)) \oplus e'$ and compute the quantity:

$$s = \frac{c(m')}{c(m)} \bmod p\,. \tag{5.7}$$

Using Equation 5.5 s can be written as:

$$s = \frac{a}{b} \bmod p, \quad \begin{cases} a & = \prod_{(m'_i=1)\wedge(m_i=0)} p_i \\ b & = \prod_{(m'_i=0)\wedge(m_i=1)} p_i \end{cases} \tag{5.8}$$

Note that since $h(m' \oplus m) \leq t$, we have that a and b are strictly smaller than $(p_k)^t$. Theorem 5.13 from [FSW02] shows that given t the receiver can recover a and b efficiently using a variant of Gauss' algorithm [Val91].

Theorem 5.9 *Let $a, b \in \mathbb{Z}$ such that $-A \leq a \leq A$ and $0 < b \leq B$. Let p be some prime integer such that $2AB < p$. Let $s = a \cdot b^{-1} \mod p$. Then given A, B, s and p, a and b can be recovered in polynomial time.*

12. *i.e.* encoded and potentially corrupted

As $0 \leq a \leq A$ and $0 < b \leq B$ where $A = B = (p_k)^t - 1$ and $2AB < p$ from Equation 5.4, we can recover a and b from t in polynomial time. Then, by testing the divisibility of a and b with respect to the small primes p_i, the receiver can recover $m' \oplus m$ and eventually m.

A numerical example is given below.

Example 5.9 *Let m be the 10-bit message* 1100100111. *For $t = 2$, we let p be the smallest prime number greater than $2 \cdot 29^4$, i.e. $p = 707293$. We generate the redundancy:*

$$c(m) = 2^1 \cdot 3^1 \cdot 5^0 \cdot 7^0 \cdot 11^1 \cdot 13^0 \cdot 17^0 \cdot 19^1 \cdot 23^1 \cdot 29^1 \bmod 707293$$

$$\Rightarrow c(m) = 836418 \bmod 707293 = 129125\,.$$

As we focus on the new error-correcting code we simply omit the Reed-Muller component. The encoded message is

$$\nu(m) = \mathtt{1100100111}_2 \| \mathtt{129125}_{10}\,.$$

Let the received encoded message be $\alpha = \mathtt{1100101011}_2 \| \mathtt{129125}_{10}$. Thus,

$$c(m') = 2^1 \cdot 3^1 \cdot 5^0 \cdot 7^0 \cdot 11^1 \cdot 13^0 \cdot 17^1 \cdot 19^0 \cdot 23^1 \cdot 29^1 \bmod p$$

$$\Rightarrow c(m') = 748374 \bmod 707293 = 41081\,.$$

Dividing by $c(m)$ we get

$$s = \frac{c(m')}{c(m)} = \frac{41081}{129125} \bmod 707293 = 632842$$

Applying the rationalize and factor technique we obtain $s = \dfrac{17}{19} \bmod 707293$. It follows that $m' \oplus m =$ 0000001100. *Flipping the bits retrieved by this calculation, we recover m.*

Bootstrapping: Note that instead of using an existing code as a sub-contractor for protecting $c(m)$, the sender may also recursively apply the new scheme described above. To do so consider $c(m)$ as a message, and protect $\bar{c} = c(c(\cdots c(c(m))))$, which is a rather small value, against accidental alteration by replicating it $2t + 1$ times. The receiver will use a majority vote to detect the errors in $\bar{c}$.

5.3.2.2.1 Performance of the New Error-Correcting Code for $\gamma = 2$.

Lemma 5.10 *The bit-size of $c(m)$ is:*

$$\log_2 p \simeq 2 \cdot t \log_2(k \ln k). \tag{5.9}$$

Proof: From Equation 5.4 and the Prime Number Theorem [13].

□

The total output length of the new error-correcting code is therefore $\log_2 p$, plus the length k of the message m.

C' outperforms the initial error correcting code C if, for equal error capacity t and message length k, it outputs a shorter encoding, which happens if $n' < n$, keeping in mind that both n and n' depend on k.

Corollary 5.11 *Assume that there exists a constant $\delta > 1$ such that, for k large enough, $n(k) \geq \delta k$. Then for k large enough, $n'(k) \leq n(k)$.*

13. $p_k \simeq k \ln k$.

Proof: Let k be the size of m and k' be the size of $c(m)$.

We have $n'(k) = k + n(k')$, therefore

$$n(k) - n'(k) = n(k) - (k + n(k')) \geq (\delta - 1)k - n(k').$$

Now,

$$(\delta - 1)k - n(k') \geq 0 \Leftrightarrow (\delta - 1)k \geq n(k').$$

But $n(k') \geq \delta k'$, hence

$$(\delta - 1)k \geq \delta k' \Rightarrow k \geq \frac{k'\delta}{(\delta - 1)}.$$

Finally, from Lemma 5.10, $k' = O(\ln \ln k!)$, which guarantees that there exists a value of k above which $n'(k) \leq n(k)$.

□

In other terms, any correcting code whose encoded message size is growing linearly with message size can benefit from the described construction.

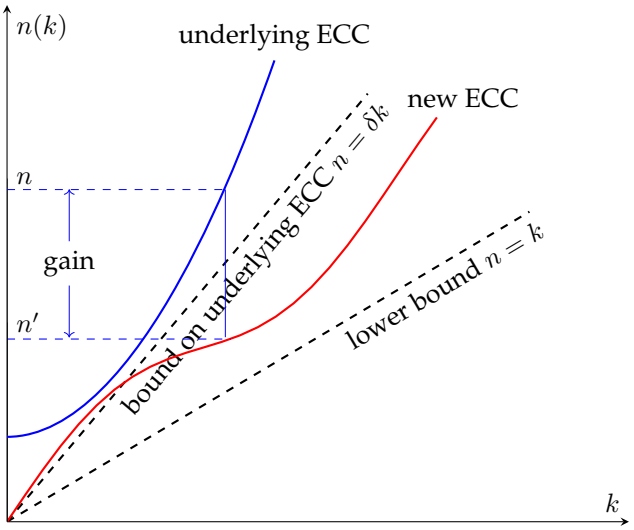


Figure 5.8: Illustration of Corollary 5.11. For large enough values of k, the new ECC uses smaller codewords as compared to the underlying ECC.

Expansion Rate: Let k be the length of m and consider the bit-size of the corresponding codeword as in Equation 5.9. The expansion rate ρ is:

$$\rho = \frac{\|m\|\mu(c(m))\|}{\|m\|} = \frac{k + \|\mu(c(m))\|}{k} = 1 + \frac{\|\mu(c(m))\|}{k}. \tag{5.10}$$

5.3.2.2.2 Reed-Muller Codes. We illustrate the idea with Reed-Muller codes. Reed-Muller (R-M) codes are a family of linear codes. Let $r \geq 0$ be an integer, and $N = \log_2 n$, it can apply to messages of size

$$k = \sum_{i=1}^{r} \binom{N}{i}. \tag{5.11}$$

Such a code can correct up to $t = 2^{N-r-1} - 1$ errors. Some examples of $\{n, k, t\}$ triples are given in Table 5.4. For instance, a message of size 163 bits can be encoded as a 256-bit string, among which up to 7 errors can be corrected.

Table 5.4: Examples of length n, dimension k, and error capacity t for Reed-Muller code.

n	16	64	128	256	512	2048	8192	32768	131072
k	11	42	99	163	382	1024	5812	9949	65536
t	1	3	3	7	7	31	31	255	255

Table 5.5: (n, k, t)-codes generated from Reed-Muller by our construction.

n'	638	7860	98304
k	382	5812	65536
$c(m)$	157	931	9931
$\mathrm{RM}(c(m))$	256	2048	32768
t	7	31	255

To illustrate the benefit of our approach, consider a 5812-bit message, which we wish to protect against up to 31 errors.

A direct use of Reed-Muller would require $n(5812) = 8192$ bits as seen in Table 5.4. Contrast this with our code, which only has to protect $c(m)$, that is 931 bits as shown by Equation 5.9, yielding a total size of $5812 + n(931) = 5812 + 2048 = 7860$ bits.

Other parameters for the Reed-Muller primitive are illustrated in Table 5.5.

Table 5.5 shows that for large message sizes and a small number of errors, our error-correcting code slightly outperforms Reed-Muller code.

5.3.2.2.3 The Case $\gamma > 2$. The difficulty in the case $\gamma > 2$ stems from the fact that a binary error in a γ-base message will in essence scramble all digits preceding the error. As an example,

$$\underline{12200210}12202012010011120202_3 + 2^{30} = \underline{12200210}221120001122201101 10_3$$

Hence, unless $\gamma = 2^\Gamma$ for some Γ, a generalization makes sense only for channels over which transmission uses γ symbols. In such cases, we have the following: a k-bit message m is pre-encoded as a γ-base κ-symbol message m'. Here $\kappa = \lceil k/\log_2 \gamma \rceil$. Equation 5.4 becomes:

$$2 \cdot p_\kappa^{2t(\gamma-1)} < p < 4 \cdot p_\kappa^{2t(\gamma-1)}$$

Comparison with the binary case is complicated by the fact that here t refers to the number of *any* errors regardless their semiologic meaning. In other words, an error transforming a 0 into a 2 counts exactly as an error transforming 0 into a 1.

Example 5.10 *As a typical example, for $t = 7$, $\kappa = 10^6$ and $\gamma = 3$, $p_\kappa = 15485863$ and p is a 690-bit number.*

For the sake of comparison, $t = 7$, $k = 1584963$ (corresponding to $\kappa = 10^6$) and $\gamma = 2$, yield $p_k = 25325609$ and a 346-bit p.

5.3.2.3 Improvement Using Smaller Primes

The construction described in the previous section can be improved by choosing a smaller prime p, but comes at a price; namely decoding becomes only heuristic. T

Parameter Generation: The idea consists in generating a prime p smaller than before. Namely, we generate a p satisfying :

$$2^u \cdot p_k^t < p < 2^{u+1} \cdot p_k^t \tag{5.12}$$

for some small integer $u \geq 1$.

Encoding and Decoding: Encoding remains as previously. The redundancy $c(m)$ being approximately half as small as the previous section's one, we have :

$$s = \frac{a}{b} \mod p, \quad \begin{cases} a & = \prod_{(m'_i=1)\wedge(m_i=0)} p_i \\ b & = \prod_{(m'_i=0)\wedge(m_i=1)} p_i \end{cases} \tag{5.13}$$

and since there are at most t errors, we must have :

$$a \cdot b \leq (p_k)^t \tag{5.14}$$

We define a finite sequence $\{A_i, B_i\}$ of integers such that $A_i = 2^{u \cdot i}$ and $B_i = \lfloor 2p/A_i \rfloor$. From Equations 5.12 and 5.24 there must be at least one index i such that $0 \leq a \leq A_i$ and $0 < b \leq B_i$. Then using Theorem 5.13, given A_i, B_i, p and s, the receiver can recover a and b, and eventually m.

The problem with that approach is that we lost the guarantee that $\{a, b\}$ is unique. Namely we may find another $\{a', b'\}$ satisfying Equation 5.13 for some other index i'. We expect this to happen with negligible probability for large enough u, but this makes the modified code heuristic (while perfectly implementable for all practical purposes).

5.3.2.3.1 Performance.

Lemma 5.12 *The bit-size of $c(m)$ is:*

$$\log_2 p \simeq u + t \log_2(k \ln k). \tag{5.15}$$

Proof: Using Equation 5.12 and the Prime Number Theorem.

□

Thus, the smaller prime variant has a shorter $c(m)$.

As u is a small integer (*e.g.* $u = 50$), it follows immediately from Equation 5.4 that, for large n and t, the size of the new prime p will be approximately half the size of the prime p generated in the preceding section.

This brings down the minimum message size k above which our construction provides an improvement over the bare underlying correcting code.

Note: In the case of Reed-Muller codes, this variant provides no improvement over the technique described in Section 5.3.2.2 for the following reasons: (1) by design, Reed-Muller codewords are powers of 2; and (2) Equation 5.15 cannot yield a twofold reduction in p. Therefore we cannot hope to reduce p enough to get a smaller codeword.

That doesn't preclude other codes to show benefits, but the authors did not look for such codes.

5.3.2.4 Prime Packing Encoding

It is interesting to see whether the optimization technique of [CMNS08] yields more efficient ECCs. Recall that in [CMNS08], the p_is are distributed amongst κ packs. Information is encoded by picking one p_i per pack. This has an immediate impact on decoding: when an error occurs and a symbol σ is replaced by a symbol σ', both the numerator and the denominator of s are affected by *additional* prime factors.

Let $C = (\mu, \mu^{-1}, \mathcal{M}, \mathcal{C}, \mathcal{P})$ be a t-error capacity code, such that it is possible to efficiently recover c from $\mu(c) \oplus e$ for any c and any e, where $h(e) \leq t$. Let $\gamma \geq 2$ be a positive integer.

Before we proceed, we define $\kappa := \lceil k / \log_2 \gamma \rceil$ and

$$f := f(\gamma, \kappa, t) = \prod_{i=k-t}^{k} p_{\gamma i}.$$

Parameter Generation: Let p be a prime number such that:

$$2 \cdot f^2 < p < 4 \cdot f^2 \tag{5.16}$$

Let $\hat{\mathcal{C}} = \mathcal{M} \times \mathbb{Z}_p$ and $\hat{\mathcal{P}} = (\mathcal{P} \cup \mathfrak{P}) \times \mathbb{N}$. We now construct a variant of the ECC presented in Section 5.3.2.2 from C and denote it

$$\hat{C} = \left(\nu, \nu^{-1}, \mathcal{M}, \hat{\mathcal{C}}, \hat{\mathcal{P}}\right).$$

Encoding: We define the "redundancy" of a k-bit message $m \in \mathcal{M}$ (represented as κ digits in base γ) by:

$$\hat{c}(m) := \prod_{i=0}^{\kappa-1} p_{i\gamma+m_i+1} \bmod p$$

A message m is encoded as follows:

$$\nu(m) := m \| \mu\left(\hat{c}\,(m)\right)$$

Decoding: The received information α differs from $\nu(m)$ by a certain number of bits. Again, we assume that the number of these differing bits is at most t. Therefore $\alpha = \nu(m) \oplus e$, where $h(e) \leq t$. Write $e = e_m \| e_{\hat{c}}$ such that

$$\alpha = \nu(m) \oplus e = m \oplus e_m \| \mu(\hat{c}(m)) \oplus e_{\hat{c}} = m' \| \mu(\hat{c}(m)) \oplus e_{\hat{c}}.$$

Since $h(e) = h(e_m) + h(e_{\hat{c}}) \leq t$, the receiver can recover efficiently $\hat{c}(m)$ from α. It is then possible to compute

$$s := \frac{\hat{c}(m')}{\hat{c}(m)} \bmod p = \frac{\prod_{i=0}^{\kappa-1} p_{i\gamma+m'_i+1}}{\prod_{i=0}^{\kappa-1} p_{i\gamma+m_i+1}} \bmod p.$$

$$s = \frac{a}{b} \bmod p, \quad \begin{cases} a &= \prod_{m'_i \neq m_i} p_{i\gamma+m'_i+1} \\ b &= \prod_{m_i \neq m'_i} p_{i\gamma+m_i+1} \end{cases} \tag{5.17}$$

As $h(e) = h(e_m) + h(e_{\hat{c}}) \leq t$, we have that a and b are strictly smaller than $f(\gamma,\kappa)^{2t}$. As $A = B = f(\gamma,\kappa)^{2t} - 1$, we observe from Equation 5.16 that $2AB < p$. We are now able to recover a, b, $\gcd(a,b) = 1$ such that $s = a/b \bmod p$ using lattice reduction [Val91].

Testing the divisibility of a and b by $p_1, \ldots, p_{\kappa\gamma}$ the receiver can recover $e_m = m' \oplus m$, and from that get $m = m' \oplus e_m$. Note that by construction only one prime amongst γ is used per "pack": the receiver can therefore skip on average $\gamma/2$ primes in the divisibility testing phase.

5.3.2.4.1 Performance. Rosser's theorem [Dus99, Ros38] states that for $n \geq 6$,

$$\ln n + \ln\ln n - 1 < \frac{p_n}{n} < \ln n + \ln\ln n$$

i.e. $p_n < n(\ln n + \ln\ln n)$. Hence a crude upper bound of p is

$$\begin{aligned} p &< 4f(\kappa,\gamma,t)^2 \\ &= 4\left(\prod_{i=\kappa-t}^{\kappa} p_{\gamma i}\right)^2 \\ &\leq 4\prod_{i=\kappa-t}^{\kappa} \left(i\gamma(\ln i\gamma + \ln\ln(i\gamma))\right)^2 \\ &\leq 4\gamma^{2t}\left(\frac{\kappa!}{(\kappa-t-1)!}\right)^2 (\ln\kappa\gamma + \ln\ln\kappa\gamma)^{2t} \end{aligned}$$

Again, the total output length of the new error-correcting code is $n' = k + |p|$.

Plugging $\gamma = 3$, $\kappa = 10^6$ and $t = 7$ into Equation 5.16 we get a 410-bit p. This improves over Example 5.10 where p was 690 bits long.

5.4 Backtracking-Assisted Multiplication

A number of applications require performing long multiplications in performance-restricted environments. Indeed, low-end devices such as the 68HC05 or the 80C51 microprocessors have a very limited instruction-set, very limited memory, and operations such as multiplication are rather slow: a `mul` instruction typically claims 10 to 20 cycles.

General multiplication has been studied extensively, and there exist algorithms with very good asymptotic complexity such as the Schönhage-Strassen algorithm [SS71] which runs in time $O(n \log n \log\log n)$ or the more recent Fürer algorithm [Für09], some variants of which achieve the slightly better $O(2^{3\log^* n} n \log n)$ complexity [HVDHL14]. Such algorithms are interesting when dealing with extremely large integers, where these asymptotics prove faster than more naive approaches.

In many cryptographic contexts however, multiplication is performed between a variable and a *predetermined constant*:
- During Diffie-Hellman key exchange [DH76] or El-Gamal [EG84] a constant g must be repeatedly multiplied by itself to compute $g^x \bmod p$.
- The essential computational effort of a Fiat-Shamir prover [FFS88, FFS87] is the multiplication of a subset of fixed keys (denoted s_i in [FFS87]).
- A number of modular reduction algorithms use as a building-block multiplications (in $\mathbb{N}$) by a constant depending on the modulus. This is for instance the case of Barrett's algorithm [Bar87] or Montgomery's algorithm [Mon85].

The main strategy to exploit the fact that one operand is constant consists in finding a decomposition of the multiplication into simpler operations (additions, subtractions, bitshifts) that are hardware-friendly [Ber86]. The problem of finding the decomposition with the least number of operations is known as "single constant multiplication" (SCM) problem. SCM $\in \mathcal{NP}$-complete as shown in [CS84], even if fairly good approaches exist [WH99, Avi61, DM94, DM95] for small numbers. For larger numbers, performance is unsatisfactory unless the constant operand has a predetermined format allowing for *ad hoc* simplifications.

We propose a completely different approach: the constant operand is encoded in a computation-friendly way, which makes multiplication faster. This encoding is based on linear relationships detected amongst the constant's digits (or, more generally, subwords), and can be performed offline in a reasonable time for 1024-bit numbers and 8-bit microprocessors. We use a graph-based backtracking algorithm [Knu68] to discover these linear relationships, using recursion to keep the encoder as short and simple as possible.

5.4.1 Multiplication Algorithms

We now provide a short overview of popular multiplication methods. This summary will serve as a baseline to evaluate the new algorithm's performance.

Multiplication algorithms usually fall in two broad categories: general divide-and-conquer algorithms such as Toom-Cook [Too63, Coo66] and Karatsuba [KO62]; and the generation of integer multiplications by compilers, where one of the arguments is statically known. We are interested in the case where small-scale optimizations such as Bernstein's [Ber86] are impractical, but general purpose multiplication algorithms à la Toom-Cook are not yet interesting.

We will further assume unsigned integers, and denote by w the word size (typically, $w = 8$), a_i, b_i and r_i the binary digits of a, b and r respectively:

$$a = \sum_{i=0}^{n-1} 2^{wi} a_i, \quad b = \sum_{i=0}^{n-1} 2^{wi} b_i, \quad \text{and} \quad r = a \times b = \sum_{i=0}^{2n-1} 2^{wi} r_i.$$

5.4.1.1 Textbook Multiplication

A direct way to implement long multiplication consists in extending textbook multiplication to several words. This is often done by using a MAD [14] routine.

14. An acronym standing for "Multiply Add Divide"

A MAD routine takes as input four n-bit words $\{x, y, c, \rho\}$, and returns the two n-bit words c', ρ' such that $2^n c' + \rho' = x \times y + c + \rho$. We write

$$\{c', \rho'\} \leftarrow \text{MAD}(x, y, c, \rho).$$

If such a routine is available then multiplication can be performed in n^2 MAD calls using Algorithm 19. The MIRACL big number library [Cer] provides such a functionality.

Algorithm 19: MAD-based computation of $r = a \times b$.

Input: $a, b \in \mathbb{N}$.
Output: $r \in \mathbb{N}$ such that $r = a \times b$.

```
for i ← 0 to 2n − 1 do
    r_i ← 0
end for
for i ← 0 to n − 1 do
    c ← 0
    for j ← 0 to n − 1 do
        {c, r_{i+j}} ← MAD(a_i, b_j, c, r_{i+j})
    end for
    r_{i+n} ← c
end for
return r
```

This approach is unsatisfactory: it performs more computation than often needed. Assuming a constant-time MAD instruction, Algorithm 19 runs in time $O(n^2)$.

5.4.1.2 Karatsuba's Algorithm

Karatsuba [KO62] proposed an ingenious divide-and-conquer multiplication algorithm, where the operands a and b are split as follows:

$$r = a \times b = (2^L \bar{a} + \underline{a}) \times (2^L \bar{b} + \underline{b}),$$

where typically $L = nw/2$. Instead of computing a multiplication between long integers, Karatsuba performs multiplications between shorter integers, and (virtually costless) multiplication by powers of 2. Karatsuba's algorithm is described in Algorithm 20.

Algorithm 20: Karatsuba's algorithm to compute $r = a \times b$.

Input: $a, b \in \mathbb{Z}$.
Output: $r \in \mathbb{Z}$ such that $r = a \times b$.

```
u = ā × b̄
v = a̲ × b̲
w = (ā + a̲)(b̄ + b̲) − u − v
r = 2^{2L} × u + 2^L × w + v
return r
```

This approach is much faster than naive multiplication – on which it still relies for multiplication between short integers – and runs [15] in $\Theta(n^{\log_2 3})$.

5.4.1.3 Bernstein's Multiplication Algorithm

When one of the operands is constant, different ways to optimize multiplication exist. Bernstein [Ber86] provides a branch-and-bound algorithm based on a cost function.

15. When repeated recursively.

Table 5.6: An example showing how linear relationships between individual words are encoded and interpreted.

step	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	meaning	reg1	reg2	reg3
0				1		=		1			$a_5 \leftarrow a_3 + a_7$	a_5	a_3	a_7
1					=	2					$a_4 \leftarrow a_5 + a_5$	a_5	a_4	a_7
2	=				-1			1			$a_0 \leftarrow a_7 - a_4$	a_0	a_4	a_7
3					2					=	$a_9 \leftarrow a_4 + a_4$	a_0	a_4	a_9
4	1	=								-1	$a_1 \leftarrow a_0 - a_9$	a_0	a_1	a_9
5		1							=	1	$a_8 \leftarrow a_1 + a_9$	a_8	a_1	a_9
6		1	=						1		$a_2 \leftarrow a_1 + a_8$	a_8	a_1	a_2
7			2				=				$a_6 \leftarrow a_2 + a_2$	a_8	a_6	a_2

The minimal cost, and an associated sequence, are found by exploring a tree, possibly using memoization to avoid redundant searches. More elaborate pruning heuristics exist to further speedup searching. The minimal cost path produces a list of operations which provide the result of multiplication.

Because of its exponential complexity, Bernstein's algorithm is quickly overwhelmed when dealing with large integers. It is however often implemented by compilers for short (32 to 64-bit) constants.

5.4.2 The Proposed Algorithm

5.4.2.1 Intuitive Idea

An alternative representation of the constant operand a: express some a_i as a linear combination of other a_js with small coefficients. Reconstructing $b \times a$ from the values of the $b \times a_j$ only becomes easy.

The more linear combinations we can find, the less multiplications we need to perform. Our algorithm tries to find the longest sequence of linear relationships between the digits of a. We call this sequence's length the *coverage* of a.

Yet another performance parameter is the number of registers used by the multiplier. Ideally at any point in time two registers holding intermediate values should be used. This is not always possible and depends on the digits of a.

In the next table we express a as a subset of words $A \in \{a_0, \ldots, a_{n-1}\}$ and build a sparse table U where $U_{i,j} \in \{-1, 0, 1, 2, =\}$, which encodes linear relationships between individual words.

Hence it suffices to compute $b \times a_3$ and $b \times a_7$ to infer all other $b \times a_i$ by long integer additions. Note that the algorithm only needs to allocate three $(n+1)$-word registers `reg1`, `reg2` and `reg3` to store intermediate results.

The values allowed in U can easily be extended to include more complex relationships (larger coefficients, more variables, etc.) but this immediately impacts the algorithm's performance. Indeed, the corresponding search graph has correspondingly many more branches at each node.

Operations can be performed without overflowing (*i.e.* so that results fit in a word), or modulo the word size. In the latter case, it is necessary to subtract $b \ll w$ from the result, where w is the word size, to obtain the correct result. This incurs some additional cost.

5.4.2.2 Backtracking Algorithm

Algorithm 21: macro Step(u, w).

1 $(p_{d+1,0}, p_{d+1,1}) \leftarrow (u, w)$
2 Backtrack$(d+1)$

Linear combinations amongst words of a are found by backtracking [Knu68], the pseudocode of which is given in Algorithm 23. Our implementation focuses on linear dependencies amongst 8-bit words,

Algorithm 22: macro EncodeDep(c, opcode).

if $c < 256$ **then**
  if $v_c =$ `False` **then**
    $(v_c, p_{d,2}, p_{d,3}) \leftarrow$ (`True`, c, opcode)
    Step(a, b)
    Step(a, c)
    Step(b, c)
    $v_c \leftarrow$ `False`
  end if
end if

Algorithm 23: macro Backtrack(d).

if $d > d^{\max}$ **then**
  $(d^{\max}, p^{\max}) \leftarrow (d, p)$
end if
$(a, b) \leftarrow (p_{d,0}, p_{d,1})$
for opcode $\in \mathcal{C}$ **do**
  EncodeDep(opcode(a, b), opcode)
end for

Algorithm 24: Main backtracking program.

Input: $A = \sum_{i=0}^{N-1} 256^i A_i$.
Output: $U_{i,j}$

// Initialization
for $i = 0$ to 255 **do**
  $v_i \leftarrow$ `True`
end for
for $i = 0$ to $N - 1$ **do**
  $v_{A_i} \leftarrow$ `False`
end for
$d^{\max} \leftarrow -1$
// Backtracking
for $i = 0$ to 255 **do**
  for $j = i + 1$ to 255 **do**
    if $v_i = v_j =$ `False` **then**
      $(p_{0,0}, p_{0,1}, v_i, v_j) \leftarrow (i, j,$ `True`, `True`$)$
      Backtrack(0)
      $(v_i, v_j) \leftarrow$ (`False`, `False`)
    end if
  end for
end for
// U-matrix reconstruction
$U_{i,j} \leftarrow 0$
for $i = 0$ to 255 **do**
  $(\text{in}_1, \text{in}_2, \text{out}, \text{opcode}) \leftarrow p_i^{\max}$
  $(U_{i,\text{in}_1}, U_{i,\text{in}_2}, U_{i,\text{out}}) \leftarrow (1, 1, \text{opcode})$
end for
return $U_{i,j}$

as our main recommendation for applying the proposed multiplication algorithm is exactly an 8-bit microprocessor.

We take advantage of recursion and macro expansion (see Algorithms 21, 22 and 23) to achieve a more

compact code. In this implementation, p encodes the current depth's three registers of Table 5.6 as well as the current operation. With suitable listing, Algorithm 24 outputs a set of values being related, along with the corresponding relation. The dependencies that we take into account in our C code (given in Appendix E) don't go beyond depth 2. Thus, the corresponding operations are $\mathcal{C} = \{+, -, \times 2\}$. We also add these operations performed modulo 256, to obtain more solutions. The alternative to this approach is to consider a bigger depth, which naturally leads to more possibilities.

Our program takes as an input an integer p that represents the percentage of a being covered (*i.e.* the coverage is $p/100$ times the length of the a). In a typical lightweight scenario, a 128-byte number is involved in the multiplication process. Our software attempts to backtrack over a coverage-related number of values out of 256. It follows immediately that at most a 50% coverage would be required for performing such a multiplication (as byte collisions are likely to happen).

The program takes as parameter the list of bytes of a. If some bytes appear multiple times, it is not necessary to re-generate each of them individually: generation is performed once, and the value is cloned and dispatched where needed.

Note that if precomputation takes too long, the list of a_i can be partitioned into several sub-lists on which backtrackings are run independently. This would entail as many initial multiplications by the online multiplier but still yield appreciable speed-ups.

5.4.2.3 Multiplication Algorithm

Algorithm 25: Virtual Machine

Input: b, instr $= (\text{opcode}, i, j, t, p)_k, R$
Output: r

$r \leftarrow 0$
foreach $(i, v) \in R$ **do**
  $\texttt{reg}[i] \leftarrow v \times b$
  $\mathsf{PlaceAt}(v, \texttt{reg}[i])$
end foreach
foreach $(\text{opcode}, i, j, t, p) \in$ instr **do**
  $\texttt{reg}[t] \leftarrow \text{opcode}(\texttt{reg}[i], \texttt{reg}[j])$
  $\mathsf{PlaceAt}(p, t)$
end foreach
return r

With the encoding of a generated by Algorithm 24, it is now possible to implement multiplication efficiently.

To that end we make use of a specific-purpose multiplication virtual machine (VM) described in Algorithm 25. The VM is provided with instructions of the form

```
opcode t, i, j, p
```

that are extracted offline from U. Here, `opcode` is the operation to perform, `i` and `j` are the indices of the operands, `t` is the index of the result, and $\texttt{p} \leftarrow w \times \texttt{t}$ is the position in r where to place the result, w being the word size. The value of `p` is pre-computed offline to allow for a more efficient implementation.

We store the result in a $2n$-byte register initialized with zero. We also make use of a long addition procedure $\mathsf{PlaceAt}(\texttt{p}, i)$ which "places" the contents of the $(n+1)$-byte register $\texttt{reg}[i]$ at position `p` in r. PlaceAt performs the addition of register $\texttt{reg}[i]$ starting from an offset `p` in r, propagating the carry as needed.

Finally, we assume that the list $R = (i, v)_k$ of root nodes (position and value) of U is provided.

After executing all the operations described in U, Algorithm 25 has computed $r \leftarrow a \times b$.

Table 5.7: Performance on a 68HC05 clocked at 5 MHz.

	Time	RAM	Code Size
Usual Algorithm	188 ms	395 bytes	1.1 kilobytes
New Algorithm	72 ms	663 bytes	1.7 kilobytes

Karatsuba Multiplication Using the notations of Algorithm 20 one can see that in settings where a is a constant, the numbers u, v, w all result from the multiplication of $\bar{b}, \underline{b}$ and $\bar{b}+\underline{b}$ (which are variable) by $\bar{a}, \underline{a}$ and $\bar{a}+\underline{a}$ (which are constant). Hence our approach can independently be combined with Karatsuba's algorithm to yield further improvements.

5.4.3 Performance

The algorithm has an offline step (backtracking) and an online step (multiplication), which are implemented on different devices.

The offline step is naturally the longest; its performance is heavily dependent on the digit combination operations allowed and on how many numbers are being dealt with. More precisely, results are near-instant when dealing with 64 individual bytes and operations $\{+, -, \times 2\}$. It takes much longer if operations modulo 256 are considered as well, but this gives a better coverage of a, hence better *online* results. That being said, modulo 256 operations are slightly less efficient than operations over the integers ($\simeq 1.5$ more costly), since they require a subtraction of b afterwards.

Table 5.7 provides comparative performance data for a multiplication by the processed constant $\lfloor \pi 2^{1024} \rfloor$. Backtracking this constant took 85 days on an Altix UV1000 cluster.

Antoine Joux [Jou] rightly noted that for long operands (*e.g.* 2048 bits) 255 successive additions of b to itself cover all the digits of a and that, in essence, the vast majority of these values will be actually used. This will avoid backtracking altogether. While 2048-bit operands are rarely used in lightweight implementations, Joux's idea allows to further improve the backtracking's efficiency by adding to $\mathcal{C}$ the additional opcode "`PlusOne`" consisting in adding a to a register.

As a final remark, note that one can also reverse the idea and generate a key by which multiplication is easy. This can be done by progressively picking VM operations until an operand (key) with sufficient entropy is obtained. While this is not equivalent to randomly selecting keys, we conjecture that, in practice, the existence of linear relations [16] between key bytes should not significantly weaken public-key implementations.

16. These linear relations are unknown to the attacker.

5.5 Regulating the Pace of von Neumann Extractors

In a celebrated paper published in 1951 [vN51], von Neumann presented a simple procedure allowing to correct the bias of random sources. Consider a biased binary source $\mathcal{S}$ emitting 1s with probability p and 0s with probability $1-p$. A von Neumann extractor $\mathcal{C}$ queries $\mathcal{S}$ twice to obtain two bits a, b until $a \neq b$. When $a \neq b$ the extractor outputs a.

Because $\mathcal{S}$ is biased, $\Pr[ab = 11] = p^2$ and $\Pr[ab = 00] = (1-p)^2$, but $\Pr[ab = 01] = \Pr[ab = 10] = p(1-p)$. Hence $\mathcal{C}$ emits 0s and 1s with equal probability.

Cryptographic hardware is usually synchronous. Algorithms such as stream ciphers, block ciphers or even modular multipliers usually run in a number of clock cycles which is independent of the operands' values. Feeding such HDL blocks with the inherently irregular output of $\mathcal{C}$ frequently proves tricky [17].

We propose a new building block called Pace Regulator (denoted $\mathcal{R}$). $\mathcal{R}$ is inserted between the randomness consumer $\mathcal{F}$ and $\mathcal{C}$ to regulate the pace at which random bits reach $\mathcal{F}$ (Figure 5.9).

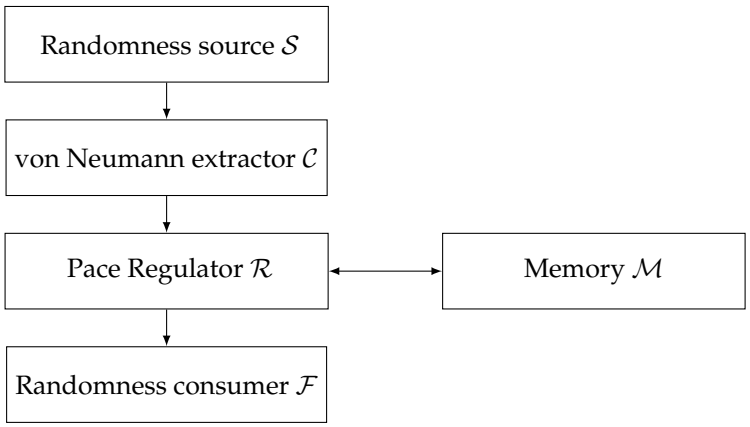


Figure 5.9: Source correction and regulation.

5.5.1 Model and Assumptions

In all generality we have at one end of a chain a generator $\mathcal{G}$ (here, $\mathcal{G} = \mathcal{S} \circ \mathcal{C}$) that outputs a stream of objects, continuously but at a varying rate. Objects are denoted by $a_1, a_2, \ldots$. At the other end, there is a client $\mathcal{F}$ that we wish to feed objects in a timely fashion, *i.e.* at a near-constant rate.

We wish to design a state machine $\mathcal{R}$ that sits between $\mathcal{G}$ and $\mathcal{F}$, and turns the erratic output of $\mathcal{G}$ into a tame inflow for $\mathcal{F}$. To this end, $\mathcal{R}$ may employ a temporary limited storage $\mathcal{M}$. The setting is illustrated in Figure 5.10.

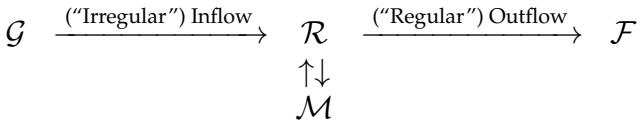


Figure 5.10: Problem: Design $\mathcal{R}$ so that the outflow from $\mathcal{R}$ to $\mathcal{F}$ is as smooth as possible, despite the outflow from $\mathcal{G}$ being variable.

The output rate of $\mathcal{G}$ is governed by a probability distribution: an a_i is emitted every t time units, where t is a random variable with probability distribution T.

We make the following important assumptions:

17. A similar problem is met when RSA primes must be injected into mobile devices on an assembly line. Because the time taken to generate a prime is variable, optimizing a key injection chain is not straightforward.

(H1) T is compactly supported, *i.e.* there exists a maximum possible waiting time $t_{\max}$ and a minimum waiting time $t_{\min}$ which we know.

(H2) The a_is produced by $\mathcal{G}$ do not expire, their order does not matter, and they can be stored in $\mathcal{M}$ indefinitely if needed. Hence we can think of $\mathcal{M}$ as a stack of size m.

(H3) Interaction between $\mathcal{R}$ and $\mathcal{M}$ is much faster than waiting times and can for all practical purposes be considered instantaneous.

5.5.2 Generic Regulator Description

Informally, the idea behind the regulator concept is that we can use $\mathcal{M}$ to store some a_js, which we may later insert between $\mathcal{G}$'s outputs if $\mathcal{G}$ takes "too long". We cannot store infinitely many objects, and conversely we cannot fill $\mathcal{G}$'s gaps if $\mathcal{M}$ is depleted. Therefore we must determine when to store objects we receive, and when to emit stored objects.

Mathematically, let $\mu > 0$ be some pivot value to be determined later. We assume that $\mathcal{R}$ maintains a timer, so that we know the time t_i elapsed between the emission of a_{i-1} and a_i. We then treat a_is as follows:

- $t_i < \mu$: a_i is "early". Store a_i in $\mathcal{M}$ for later use.
- $t_i = \mu$: a_i is "timely". Output a_i immediately to $\mathcal{F}$.
- If μ time units have elapsed, and still no a_i has been received from $\mathcal{G}$ ("late"), we fetch an a_j from $\mathcal{M}$, send a_j to $\mathcal{F}$, and act as if a_j were just received (*i.e.* a_i is given μ additional time units to arrive: $t_i \leftarrow t_i - \mu$).

Therefore if μ is properly chosen, so that $\mathcal{M}$ never overflows and is never empty, $\mathcal{R}$ outputs one a_i every μ.

Furthermore, we wish $\mathcal{R}$ to be as simple as possible, and in this work consider that $\mathcal{R}$ is an event-driven state machine having access to the following primitives:

- $\mathsf{Push}(a)$ pushes a on the stack $\mathcal{M}$.
- $\mathsf{Pop}()$ pops an object a from the stack and emits it to $\mathcal{F}$.
- $\mathsf{Stack}()$ returns the number of objects currently stored in $\mathcal{M}$.
- $\mathsf{Signal}(t)$ registers an event EventSig (see below) to be called after time t has elapsed.

The events are:

- EventSig is called when time t has elapsed since the call of $\mathsf{Signal}(t)$.
- $\mathsf{ObjIn}(a)$ is called when an object is received from $\mathcal{G}$.
- $\mathsf{Setup}(x)$ is called once at initialization.
- $\mathsf{Error}()$ is called upon errors.

$\mathcal{R}$ is inactive between events: it is entirely characterized by describing what it does when events occur.

5.5.2.1 Generic Regulator

The regulator's functionality is achieved by using the event handlers described in Algorithms 26, 27 and 28. For the sake of simplicity, we allow $\mathcal{R}$ to use a single global variable s for its operation which we do not count as part of $\mathcal{M}$ in the following discussion. We purposely leave the error handler unspecified.

Algorithm 26: $\mathsf{Setup}()$

```
1 s ← t_max
2 Signal(s)
```

The main question thus is how to choose the function μ appropriately. For $\mathcal{M}$ to be neither empty nor overflow in the long term, it is necessary that the number of a_js being stored ("early a_js") and the number of a_js being fetched ("late a_js") balance each other.

Algorithm 27: ObjIn(a)

```
X ← Stack()
if X < |M| then
    Push(a)
    else
        Error()
    end if
end if
```

Algorithm 28: EventSig

```
X ← Stack()
if 0 < X then
    s ← μ(X)
    Pop()
    else
        Error()
    end if
end if
Signal(s)
```

5.5.3 The Median Regulator

One way to achieve this balance is to choose $\mu(X) = \mu_M$ such that $T(t < \mu_M) = T(t > \mu_M)$, which is exactly the definition of the median. Hence, we can set

$$\mu_M := t_{1/2} = \text{Median}(t)\,. \tag{5.18}$$

Implementing the generic regulator with this choice of μ yields the *median regulator*. Note that the sample median could be estimated from the data and used here, instead of the theoretical median (if unknown).

Equation 5.18 is not a *sufficient* condition: it may be that while being zero *on average*, the amount of a_j stored in $\mathcal{M}$ wanders around. Indeed, there is a $1/2$ probability to get an early (resp. late) a_i [18], so that the population X_k of $\mathcal{M}$ undergoes a random walk. We have

$$\lim_{k\to\infty} \frac{\mathbb{E}\left(|X_k - \frac{m}{2}|\right)}{\sqrt{k}} = \sqrt{\frac{2}{\pi}} \quad \Rightarrow \quad \left|X_k - \frac{m}{2}\right| \approx \sqrt{k}\,.$$

Therefore, on average, this regulator reaches an error state after receiving $\sqrt{m}$ a_is. $\mathcal{M}$ could be chosen so that $m \approx k^2$ where k is the maximal number of packets that we wish to process. However this limitation is unsatisfactory and we will get rid of it.

5.5.4 Memory-Variance Trade-Off: Adaptive Regulators

The key observation is that Equation 5.18 is not a *necessary condition* either: all that is required is really that $\mathbb{E}(\mu) = t_{1/2}$. Now we may be smarter and adjust the value of μ to the moment's needs. Indeed, if we are about to use too much memory, then decreasing μ would result in more a_js being labelled "late", and we would start emptying $\mathcal{M}$. If on the contrary $\mathcal{M}$ is getting dangerously empty, we may increase μ so that more a_js become "early", and start repopulating $\mathcal{M}$. Note that we may vary μ slowly or quickly over time, this variation being itself irrelevant to the statistical analysis.

Of course, such a strategy incurs a non-zero variance in the outflow, but at this price we may lower the size of $\mathcal{M}$. More precisely, for any given memory capacity $m = |\mathcal{M}|$ and input-time distribution T, we

18. In other term, we consider that the probability of getting a timely a_i is negligible.

want to construct an $\mathcal{R}$ whose output-time distribution T'_m is such that

$$\begin{array}{ll} \lim_{m\to\infty} & \mathrm{Var}(T'_m) = 0 \\ \lim_{m\to 0} & \mathrm{Var}(T'_m) = \mathrm{Var}(T) \\ & \mathrm{Var}(T'_m) \leq \mathrm{Var}(T) \end{array}$$

This is of course the ideal case and the further question now becomes: How do we modulate μ at any given moment in time, to achieve this?

Let X denote the occupation of $\mathcal{M}$ at a given point in time. If $X = 0$ then we *must* take in new a_is, and we cannot output any more a_js, therefore we have no choice but to set $\mu \leftarrow t_{\text{max}}$. Conversely, if $X = m$ then we must empty the queue and set [19] $\mu \leftarrow t_{\text{min}}$. We already saw that if $X = m/2$ the best choice is the neutral $\mu \leftarrow t_{1/2}$.

We wish to interpolate and describe the function $\mu(X)$ that is such that

$$\mu(0) = t_{\text{max}}, \qquad \mu(m/2) = t_{1/2}, \qquad \mu(m) = t_{\text{min}}$$

There are several ways to do so.

5.5.4.1 Lagrange Regulator

Take for instance Lagrange interpolation polynomials: let

$$\begin{aligned} a &= \frac{2}{m^2}\left(t_{\text{max}} + t_{\text{min}} - 2t_{1/2}\right) \\ b &= \frac{1}{m}\left(t_{\text{max}} + 3t_{\text{min}} - 4t_{1/2}\right) \\ c &= t_{\text{max}} \end{aligned}$$

Then we can take

$$\mu_L(X) := aX^2 + bX + c.$$

In the special case where $T = \mathrm{Uniform}(A, 3A)$, we have $\mu_L(X) = (3 - 2X/m)A$.

5.5.4.2 Distributional Regulator

The main interest of the Lagrange Regulator is its simplicity. However, there is no reason to consider that the choice of a μ polynomial in X is optimal. Let F_t be the cumulative distribution function $F_t(y) := T(t \leq y)$ and consider its inverse F_t^{-1}. We define the distributional regulator as

$$\mu_D(X) := F_t^{-1}\left(1 - \frac{X}{m}\right).$$

Observe that we have

$$\begin{aligned} \mu_D(0) &= F_t^{-1}(1) = t_{\text{max}} \\ \mu_D\left(\frac{m}{2}\right) &= F_t^{-1}\left(\frac{1}{2}\right) = t_{1/2} \\ \mu_D(m) &= F_t^{-1}(0) = t_{\text{min}} \end{aligned}$$

This regulator assumes a complete knowledge of t's distribution, but provides the best results in the sense that it minimizes the variance of $\mathcal{R}$'s output. In the special case where $T = \mathrm{Uniform}(A, 3A)$, we have

$$\mu_D(X) := F_t^{-1}\left(1 - \frac{X}{m}\right) = A + 2A\left(1 - \frac{X}{m}\right) = \left(3 - 2\frac{X}{m}\right)A = \mu_L(X)$$

that is, we get the exact same result as the Lagrange Regulator.

19. We do not set $\mu \leftarrow 0$ or any lower value for two reasons: first $\mathcal{R}$ would empty its whole stack immediately, which is not the intended behaviour; and second this makes interpretation and analysis harder.

5.5.5 Parameters for the von Neumann Extractor

We can compute exactly the distribution T for the von Neumann extractor if $\mathcal{S}$ outputs one random value every δ units of time. In that case, one couple is generated every 2δ, and this couple has a probability $2p(1-p)$ to be accepted. Each couple is generated independently from others, so that the probability of k successive rejections is $(1-2p(1-p))^k$. Let $\epsilon = 2p^2 - 2p + 1$, we have $0 < \epsilon < 1$ and

$$T(2k\delta) = \epsilon^k(1-\epsilon).$$

Observe that T is *not* compactly supported, as for any $t > 0$ we have $T(t) > 0$. However we can define a cut-off value above which event probability becomes negligible, *i.e.* $T(t) < 2^{-N}$ for some $N \in \mathbb{N}$. This gives

$$k_{\max} = -\frac{N - \log_2(1-\epsilon)}{\log_2(\epsilon)} \Rightarrow t_{\max} = -2\delta\frac{N - \log_2(1-\epsilon)}{\log_2(\epsilon)}$$

the minimum is $t_{\min} = 0$, and the median is computed from the cumulative probability

$$\sum_{k=0}^{n} T(2k\delta) = \sum_{k=0}^{n} \epsilon^k(1-\epsilon) = 1 - \epsilon^{n+1}$$

so that $k_{1/2} = -\frac{1}{\log_2 \epsilon} - 1$, hence

$$t_{1/2} = -2\delta\left(\frac{1}{\log_2 \epsilon} - 1\right)$$

Example 5.11 *Assume $\delta = 1$ and $N = 80$, we have the following parameters for different biases p:*

p	ϵ	t_{min}	$t_{1/2}$	t_{max}
$1/2$	$1/2$	0	4	162
$1/4$	$5/8$	0	4.95	241
$1/32$	$481/512$	0	24.19	1866

5.5.6 Experimental Results

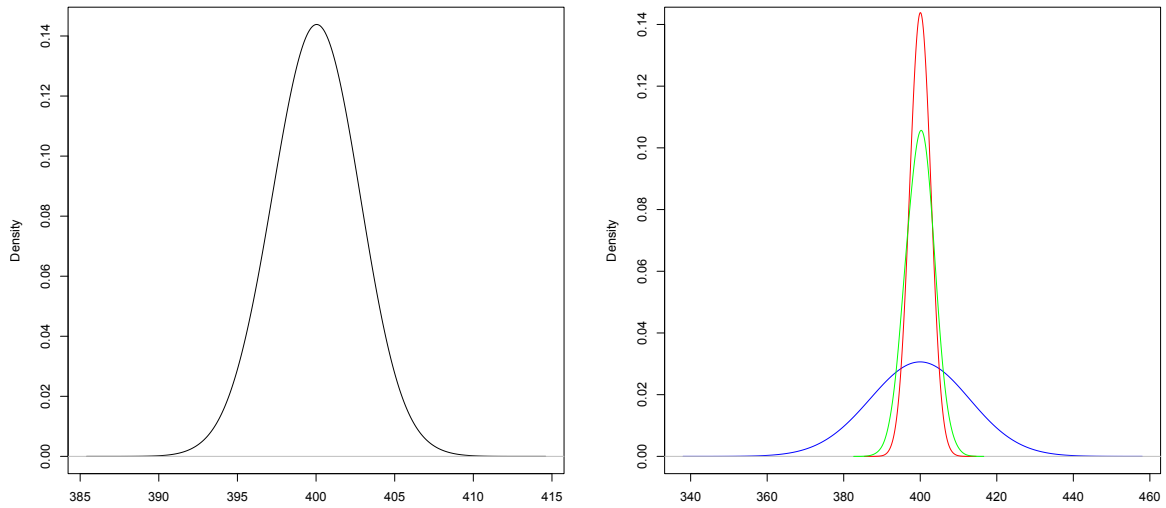


Figure 5.11: Left: Steady-state output distribution of a Lagrange regulator, with input distribution $T = \text{Uniform}(200, 600)$ and $m = 1000$. The distribution peaks at $\mu' = 400.0$, and is contained in $[390, 410]$. Compare to the input distribution ($\mu = 400, \sigma = 115.4$). Average memory usage is $500 = m/2$. Right: same thing with $m = 100$ (blue), 500 (green) and 1000 (red).

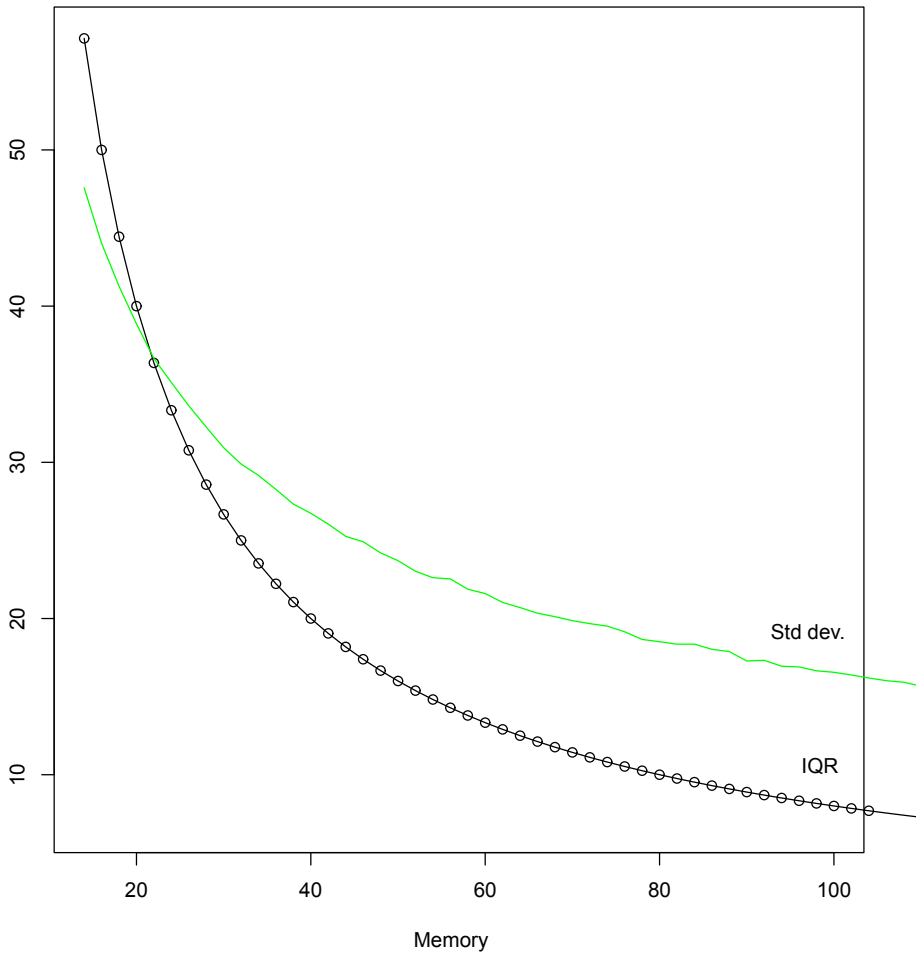


Figure 5.12: Steady-state IQR (black, circled) and standard deviation (green) as a function of m, for the same parameter set as in Figure 5.11. Both IQR and standard deviation get lower for larger values of m, and reach a minimal nonzero value; $\log \log \mathrm{IQR}$ is almost linear, with a slope of -0.008.

To test our regulator we implemented a simulation in Python. The simulation is event-driven: only a_i reception and emission are considered, which allows for an exact solution (in particular, there is no timer involved). a_i generation by $\mathcal{G}$ is simulated by inverse sampling of a given distribution. In the simulation we assume that this distribution is known, and we implement the corresponding Lagrange regulator. The source code is provided in F.

We choose a certain amount of memory m and run the simulation for $n \gg m$ objects. The output distribution is then measured.

After some warming-up time (which is of the order of $m/2$), the output distribution reaches a steady state peaked around a central value $\mu' \approx \mu$. The variance of this distribution is *much smaller* than the input variance and a larger memory m results in a narrower distribution.

5.5.6.1 Uniform Input Distribution

Figure 5.11 shows the steady-state distribution of a Lagrange regulator applied to a uniform generator. Memory usage X fluctuates around $m/2$. Figure 5.12 shows the evolution of variance and interquartile range (IQR) as a function of m.

Statistical dispersion around μ' decreases quickly as m increases: $\log\log \mathrm{IQR}$ decreases almost linearly with m. Both standard deviation and IQR reach a minimum value. IQR decreases faster than standard deviation, which yields a distribution with higher kurtosis as m increases. These observations are consistent across various parameter choices.

5.5.6.2 Cut-off Geometric Input Distribution

The output times of the von Neumann extractor follow a geometric distribution (*cf.* 5.5.5). Since this distribution is *not* compactly supported, we define a cut-off value $t_{\max}$.

We use the `random.geometric` function from `numpy` to automatically generate sequence of appropriately distributed t_is, with a cut-off at 2^{80} for the distributional regulator.

Results are similar to the uniform case, but memory usage is higher on average because of the input distribution's large tail. The cut-off incurs a non-zero (albeit negligible) failure probability, that must be dealt with: When an exceptionally large delay occurs, the degraded operation simply consists in outputting the late object as soon as it arrives.

5.6 Fault Attacks on Projective-to-Affine Coordinate Conversion

Many papers [20] have been devoted to fault attacks on ECCs. Fault attacks usually target variables involved in the elliptic curve scalar multiplication, such as the base point coordinates or the curve parameters. In this section, we consider a different ECC fault attack type, in which fault injection targets the conversion from projective to affine coordinates that typically follows the computation of the scalar multiplication (ECSM).

Related Work: Naccache, Smart and Stern showed [NSS04] that if, for a known base point P, the scalar multiplication's result $[k]P$ is output in projective (usually Jacobian) coordinates, then information on k can be recovered. However, in real-life implementations, results are returned in affine coordinates, so the attack described in [NSS04] cannot be performed.

Given this, we describe a number of fault attacks targeting the final projective-to-affine coordinate conversion step, making it possible to retrieve information in Jacobian coordinates, and hence carry out the attack described in [NSS04].

Organisation: Section 5.6.1 briefly recalls the EC attack of [NSS04]. In addition to [NSS04], some details on the attack's feasibility on different ECSMs and on different side-channel countermeasures are given. Section 5.6.2 describes fault injection during conversion. Depending on the type of fault, we propose three different methods for recovering the Jacobian coordinates. The three methods and their cryptanalytic consequences are described in Sections 5.6.3, 5.6.4 and 5.6.5.

5.6.1 Preliminaries

We refer the reader to Section 3.4 for an algorithmic presentation of elliptic curves.

A brief overview of [NSS04] is given here. This attack is described for ECs defined over prime fields of large characteristic, but can easily be adapted, *mutatis mutandis*, to other base fields.

5.6.1.1 Faults and Failures in Cryptography

Provable security became a central problem in cryptography since the 1990s. The theoretical development of this topic has been astonishing. As computationally secure cryptosystems started being widely adopted, researchers got used to invoke flawless mathematical models, but practical implementation issues started to surface.

Real world applications imply translating into cryptographic models, hardware or software. This fact has evolved into a major concern as implementations started being shown to be vulnerable to fault and side channel attacks.

We are particularly interested in fault injection attacks. Fault attacks belong to the category of active attacks and has been showed to be powerful. The generic technique consists of injecting intentionally harmful faults into cryptographic devices, observing [AK98, BDL01] and then analysing their erroneous outputs. The variant of this method is known as Differential Fault Analysis (DFA) and has been described by Biham and Shamir [BS97].

Differential Fault Analysis (DFA) DFA is a powerful attack on cryptosystems implemented in devices such as smart cards. If the device can be made to deliver erroneous output under stress (heat, radiation, power glitches, over-clocking, etc.) then a cryptanalyst comparing correct and erroneous outputs has a dangerous entry point to the processor's internals, including keys. Thus, secret keys can be disclosed by provoking computational errors during calculation.

20. *e.g.* [BCN+04, BMM00, GK04, CJ05, FLRV08, GKT10, BBPS11, FGV11]

5.6.1.2 Overview, classification and countermeasures.

The adverse effects of faults on electronic circuits were first observed in chips subjected to radioactive exposure [BCN+04]. Radiation caused accidental memory bits flipping. During the past couple of decades, various fault injection techniques were discovered, analysed and experimented.

Fault injection attacks may change the behaviour of a component, either permanently or transiently. If an attacker is able to modify memory contents or to alter operations in a cryptographic algorithm, computation errors will surely appear. An erroneous result $\tilde{r}$ as well as a correct output r corresponding to it, usually allows attackers to obtain valuable information.

Fault attacks are a real threat, especially in the case of IoT devices, which are widely being adopted currently. The continuous growing market of smartcards, RFID tags and other security tokens stimulated academic research in this area.

Electronic circuits are prone to a wide variety of fault injection attacks, either invasive or non-invasive. Within the next paragraphs, we will shortly describe some of these attacks.

A Preliminary Step. To perform some of the attacks, a preliminary step is necessary: removing the chip package and dissolving the epoxy resin covering the chip (decapsulation).

Optical Fault Attacks. Optical faults are usually injected either using simple (photographic) white light flashes or lasers.

Light radiation attacks were initially proposed in 2002 [SA03] and used a concentrated ray of light to provoke faults. The attackers only need a camera flash, a microscope and an aluminium sheet in order to concentrate the light. This attack allows the modification of selected bits in SRAM.

Laser attacks are more powerful, as their effect is similar to white light attacks, while having in addition the advantage of precision. Precision is very important advantage, given the sizes of nowaday's chips.

EEPROM can be altered using UV light, which can be focused on the EEPROM security block cell to erase lock bitS, and render memory externally readable.

Ion beams and X-rays can also used as fault sources. The advantage of these methods is their ability to induce fault attacks without necessarily decapsulating the chip. A focused ion beam (FIB) workstation consists of a vacuum chamber with a particle gun. Gallium ions are accelerated and focused from a liquid metal cathode into a beam. Attackers can thus simplify manual probing of deep metal and poly-silicon lines using FIBs. A hole is drilled to the signal line of interest this is filled with platinum, to lift the signal to the surface, where a large probing pad is created to allow easy access and fault injection.

Power Glitches. A smartcard's power supply comes from a reader which may be malicious. In such cases, an attacker may control the target's power supply. Current variations may hence be applied to the card and, thus, cause memory faults or even changes in code execution. Finally, an attacker may modify the program counter to cause uncontrolled program jumps or computation errors in the processor.

Clock Glitches. The clock signal of smartcards is also provided by the reader. As for power glitches, attackers may alter the incoming clock signal to interfere in the target chip's behavior. Sudden clock frequency variations are easy to perform. Such glitches may cause data read errors, instruction skipping or instruction switching [ADN+10].

Temperature. Temperature variations can be maliciously exploited in three ways: Electronic devices must operate within safe temperature intervals. When a chip is operated outside the bounds of these intervals cryptanalytically exploitable processing errors may occur. A second temperature attack consists in cooling RAM to extreme temperatures to avoid data erasure after a power-off [Cona]. Finally, in some silicon technologies, long-term data storage in a RAM cell will permanently "burn" the data into the cell. The etched data can then be read using specific technical means [Gut96].

Directly Injecting Electrical Faults. An *electronic probe station* is a laboratory test equipment used for physically acquiring signals from the internal nodes of a target chip. Probe stations can also inject signals in the target circuit and thereby induce faults.

Memory Overwriting. In some ROM technologies, single bits can be overwritten using a laser. This may allow attackers to alter executable code and result in key disclosure. Researchers also reported a physical attack on EPROM in which two probe needles were used to set or clear target EPROM cells to infer their contents [AK98, WBYD00]. A correct guess (resetting a zero bit or setting a one bit) will cause no behavioral changes. A bad guess will cause an error. In both cases one secret key bit will be pulled-out. This is repeated to access all the key's bits one after the other.

Fault attacks are classified according to the following criteria:

- Spatial control: the attacker may have no control on the memory address being modified. Data modification may hence occur in a given memory interval or even at a precise bit.
- Time control: If the attacked algorithm is probabilistic [21] the attacker may not know when the fault occurred. This may foil the attack.
- The number of bits affected by the fault: the fault may affect one bit, several bits, bytes or words. In some scenarios the number of affected bits is unknown to the attacker.
- The fault's success probability may depend on a number of technical factors such as the energy level of the fault injector or the wavelength used.
- The fault's duration - faults may be transient or permanent.

Protections against fault attacks [BBKN12] belong to several categories: Sensors (temperature, voltage, frequency, light, etc.) may detect faults and halt the chip before secrets are leaked. Circuits can be shielded using wire meshes and extra metal layers. Hardware modules can be duplicated to detect processing mismatch. Finally, data can also be protected by CRCs [22] to detect modification attempts.

Typical Countermeasures. Sensors added to micro-controllers typically detect variations in the external clock, the internal clock, internal and external voltage, temperature, signal, glitches and light sensors.

Integrated circuits can be covered by programmable active shields. Also, they may be equipped with signal layers able to detect probing signal injection of decapsulation attempts.

CRC (cyclical redundancy check) modules are used for data integrity. Checking the data to see whether an error has occurred during transmission, reading or writing is already common practice.

For an overview of fault injection attacks we refer the reader to [BBKN12, JT12].

Therefore, an additional difficulty for small embedded devices is that, if left unprotected, such devices are physically prone to attacks and can be tampered with. This problem must be carefully considered when designing cryptographic algorithms. Systems using embedded devices must be designed in such a way that the compromising of a few devices will endanger the entire system. This is because an adversary may obtain secrets stored in some devices and attempt to use these stolen secrets to attack the system as a whole.

5.6.1.3 Leakage in Projective Coordinates

Naccache, Smart and Stern [NSS04] observed that if $[k]P$ is given in Jacobian coordinates it becomes possible to recover information on k. We briefly overview the way in which this is achieved.

Denote by $A_i = (X_i, Y_i, Z_i)$ the value of point A at the end of iteration i in Algorithm 2. The attacker knows the output $A_0 = (X_0, Y_0, Z_0)$ in Jacobian coordinates and the input $P = (x_P, y_P)$ in affine coordinates. The attacker will attempt to reverse the scalar multiplication process *i.e.* replace doubling by halving and replace additions of P by subtractions.

21. or if the designer purposely introduced random delays to fool the attacker
22. cyclical redundancy checks

If $k_0 = 0$, A_1 can be recovered by halving A_0. Given formula (3.5):

$$Z_0 = 2Y_1Z_1 = 2y_1Z_1^4 \;\Rightarrow\; Z_1^4 = \frac{Z_0}{2y_1}\,.$$

We need to compute a fourth root to obtain Z_1 from Z_0:
– if $p \equiv 3 \pmod 4$, then computing a fourth root is possible for half of the inputs and, when possible, this computation yields two values
– if $p \equiv 1 \pmod 4$, then computing a fourth root is possible in a quarter of the cases and yields four values
We can hence obtain X_1 and Y_1 from Z_1.

If, on the other hand, $k_0 = 1$, A_1 can be recovered by subtracting P from A and halving. P is given in affine coordinates. We denote by (X_t, Y_t, Z_t) the intermediate point between doubling (step $A \leftarrow \text{ECDBL}(A)$) and addition (step if $k_i = 1$ then $A \leftarrow \text{ECADD}(A, P)$). Given formula (3.6), we have:

$$Z_0 = (x_P Z_t^2 - X_t)Z_t \;\Rightarrow\; Z_t^3 = \frac{Z_0}{x_P - x_t}\,.$$

We need to compute a cubic root to obtain Z_t from Z_0:
– if $p \equiv 1 \pmod 3$, then extracting a cubic root is possible in a third of the cases and, when possible, this calculation yields one of three possible values
– if $p \equiv 2 \pmod 3$, then extracting a cubic root is always possible and yields a unique value
We can easily obtain X_t and Y_t from Z_t. After subtraction, the attacker must halve (X_t, Y_t, Z_t) as described previously:

$$Z_1^4 = \frac{Z_t}{2y_t}\,.$$

From this observation, the opponent can recover the least significant bit of k. Indeed, if the value $\frac{Z_0}{2y_1}$ isn't a fourth power, the opponent can immediately conclude that $k_0 = 1$. If $\frac{Z_0}{2y_1}$ is a fourth power, then the attacker must try the subtraction and halving step. If subtracting P from A_0 or halving A_t is impossible, the attacker concludes that $k_0 = 0$. If both steps are possible (which happens with non-negligible probability), the attacker cannot immediately identify k_0, but can hope to do so by backtracking, *i.e.* guessing the values of k_1, k_2, etc. and computing the corresponding intermediate points until reaching one of the previous contradictions.

Once k_0 is known, the opponent can iterate the procedure starting with k_1 and so forth to extract a few more bits of k. Note that several candidate values for Z_1 arise from the reversal process as the corresponding equations have several roots, and backtracking is usually required to determine the correct one.

[NSS04] reports experimental data on the number of recovered bits and success probabilities.

To prevent this attack, the defender should in principle output results in affine coordinates. Another possible countermeasure suggested in [NSS04] is to randomize the output, replacing (X_0, Y_0, Z_0) by $(\lambda^2 X_0, \lambda^3 Y_0, \lambda Z_0)$ for some random $\lambda \in \mathbb{F}_p^*$, which effectively avoids any possible leakage from the Jacobian representation.

As a side note, we point out that, while [NSS04] also claims that attacks are thwarted by randomly flipping the sign of Z_0, this is incorrect: just as k_1 can be recovered with significant probability even though Z_1 is only known up to a sign (by simply trying both possibilities and backtracking until a contradiction is reached), k_0 can also be recovered even when Z_0 is only known up to a sign. This observation is important in our case, as the fault attacks described hereafter retrieve Z_0^2 rather than Z_0 itself.

5.6.1.4 Leakage in Projective Coordinates in Other Representations

The attack was presented in Jacobian coordinates. The attack works in another representation if a n-th root of a value is computed during backtracking, with $n > 1$. This is the case in Projective coordinates system where a point $P = (X, Y, Z)$ corresponds the affine point $(X/Z, Y/Z)$.

5.6.1.5 Leakage in Projective Coordinates in Other ECSMs

The attack was presented with the Double-and-Add algorithm. However, in embedded system, the Double-and-Add algorithm is vulnerable to a Simple Power Analysis [Cor99]. We give in this section some details of the attack on other algorithms.

Double-and-Add Always [Cor99]. This algorithm is similar to the Double-and-Add algorithm except that a dummy addition is performed if the current bit is 0. The output coordinates are the same as the classical Double-and-Add algorithm. The attack is thus applicable.

Signed Sliding Window Method [BSS99, algorithm IV.7]. This case was described in [NSS04]. If the attacker knows the coordinates of the precomputed multiples of the base point (which is generally the case because the precomputed multiples are in affine coordinates for the sake of efficiency), then the attack applies.

Sliding Window Method [BSS99, algorithm IV.4]. The same analysis of the Signed Sliding Window method holds here: if the attacker knows the coordinates of the precomputed multiples of the base point, then the attack applies.

Signed Digit Method. This is a particular case of the Signed Sliding Window method where the size of the window is one. The attack is applicable.

Montgomery Ladder with Classical Formulæ [JY02]. The Montgomery Ladder uses an additional temporary point R_1 that is not returned, the opponent gets only the Z coordinate of R_0. The attacker cannot halve R_1 or subtract R_1 without knowing the Z coordinate of R_1. The attack does not apply.

Montgomery Ladder with Co-Z Formulæ [GJM10]. Co-Z formulæ are alternative addition formulæ with points sharing the same Z coordinates. Co-Z formulæ are given in appendix B. If the Montgomery Ladder with co-Z formulæ (algorithm 33 in appendix) is used, the attacker gets the Z coordinate of R_0, and hence the Z coordinate of R_1 because they are the same. The attack can then be applied.

An important remark is that, with co-Z formulæ, only addition and subtraction are performed. Therefore, only cubic roots are computed during backtracking. If $p \equiv 2 \pmod 3$, then extracting a cubic root is always possible and yields a unique value, so backtracking cannot be applied because every guess will yield a solution. In this case, the attack does not apply.

5.6.1.6 Applicability of the Attack in the Presence of Side-Channel Countermeasures

Below, we give some details of the attack in the presence of side-channel countermeasuress.

Random Projective Coordinates [Cor99, §5.3]. Randomizing the point $A = (x, y, 1)$ into (r^2X, r^3Y, r), with $r \in \mathbb{F}_p^*$ in algorithm 2 at the beginning of the ECSM will not thwart the attack since only the Z coordinate of the final output point is needed for the attack.

Random Curve Isomorphism [JT01]. Let φ be the isomorphism defined by

$$\varphi : E \xrightarrow{\sim} E', \begin{cases} \mathcal{O} & \to & \mathcal{O} \\ (x, y) & \to & (u^{-2}x, u^{-3}y) \end{cases}$$

where $u \in \mathbb{F}_p^*$ is random. The countermeasure consisting of computing the ECSM on the random curve E' instead of E will not thwart the attack.

The inverse isomorphism in Jacobian coordinates of the point $(u^{-2}X, u^{-3}Y, Z) \in E'^{\mathcal{J}}$ consists in multiplying Z by u^{-1}.

Let d be a scalar with $d_0 = 1$. Let $P = (x_P, y_P) \in E$ be the base point (and hence $P' = (u^{-2}x_P, u^{-3}y_P)$ is the base point of E'), $Q' = (u^{-2}X_3, u^{-3}Y_3, Z_3) = [d]P'$ and $(u^{-2}X_t, u^{-3}Y_t, Z_t) = (u^{-2}x_tZ_t, u^{-3}y_tZ_t, Z_t) \in E'^{\mathcal{J}}$ the intermediate point between doubling and addition of the last iteration of the ECSM, then

$$Z_3 = (u^{-2}x_PZ_t^2 - X_t')Z_t \;\Rightarrow\; u^{-1}Z_3 = u^{-3}Z_t^3(x_P - x_t) \;\Rightarrow (u^{-1}Z_t)^3 = \frac{u^{-1}Z_3}{x_P - x_t}\,.$$

The knowledge of $(u^{-1}Z_3), x_P$ and x_t is sufficient to recover $(u^{-1}Z_t)$. By analogy, $(u^{-1}Z_t)$ can be used to halve the point (X_t', Y_t', Z_t).

Scalar Randomization [Cor99, §5.1]. Randomization of the scalar using $d' = d + r \cdot \#E$ where r is a random element of $\mathbb{F}_p$ will not thwart the attack because the attacker can grab a few bits of the integer d' which is a solution of the ECDLP.

Point Blinding [Cor99, §5.2]. Computing $Q = [d](P + R)$ instead of $[d]P$, where R is a pseudo-random point will thwart the attack since the output coordinates of the point $Q - [d]S = (X, Y, Z)$ will depend on the unknown coordinates X_1, Y_1, Z_1 of Q and X_2, Y_2, Z_2 of $[d]S$. Moreover, the knowledge of the base point $P + R$ is needed for backtracking.

5.6.1.7 Projective-to-Affine Conversion

The following procedure converts $P = (X, Y, Z) = (xZ^2, yZ^3, Z)$ from Jacobian to affine coordinates (x, y).

$$\text{Algorithm CONVERT}(X, Y, Z) = \begin{cases} r \leftarrow Z^{-1} \\ s \leftarrow r^2 \\ x \leftarrow X \cdot s \\ t \leftarrow Y \cdot s \\ y \leftarrow t \cdot r \quad \text{return}(x, y) \end{cases} \tag{5.19}$$

5.6.2 Faults During Conversion

In standardized EC protocols, the computed points are given in affine coordinates, and hence [NSS04] does not apply. Our idea is to corrupt the conversion process, so that the faulty affine results reveal the missing Z coordinate.

Suppose that an error corrupted s just after the step $s \leftarrow r^2$ (of process (5.19)). The corrupted $s + \epsilon$ yields:

$$\tilde{x} = X(s + \epsilon) \Rightarrow \tilde{x} = x + xZ^2\epsilon \tag{5.20}$$

$$\tilde{y} = Y(s + \epsilon)r \Rightarrow \tilde{y} = y + yZ^2\epsilon \tag{5.21}$$

The next sections describe three different attacks depending on the fault's precision.

5.6.3 Large Unknown Faults

5.6.3.1 Several Faulty Results and a Correct Result

Equations (5.20) and (5.21) imply

$$\frac{\tilde{x}}{x} - 1 = Z^2\epsilon \tag{5.22}$$

$$\frac{\tilde{y}}{y} - 1 = Z^2\epsilon \tag{5.23}$$

Let $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_n)$ be a vector of n faults. Each ϵ_i satisfies an equation of the form (5.22), thus the attacker knows n numbers $u_i = Z^2 \cdot \epsilon_i \bmod p$ denoted as a vector $\mathbf{u} = (u_1, ..., u_n)$.

Let $a < 1$. Assume that $\forall i \in \{1, \ldots, n\}, \epsilon_i < p^a$. We want to recover $\boldsymbol{\epsilon}$.

Let L be the lattice generated by the vector $\mathbf{u}$ and $p\mathbb{Z}^n$ in $\mathbb{Z}^n$ and let $s = Z^{-2} \bmod p$. Since $\boldsymbol{\epsilon}$ satisfies $\boldsymbol{\epsilon} = s \cdot \mathbf{u} \bmod p$, $\boldsymbol{\epsilon}$ is a vector in L, of length $\|\boldsymbol{\epsilon}\| \lesssim p^a$.

Assume further that $g = \gcd(u_1, \ldots, u_n) = 1$. This happens with probability $\approx 1/\zeta(n) \approx 1 - 2^{-n}$, which is very close to 1. Then, we have $\mathrm{vol}(L)^{\frac{1}{n}} = [\mathbb{Z}^n : L]^{\frac{1}{n}} = p^{1-\frac{1}{n}}$. Therefore, we can recover $\boldsymbol{\epsilon}$ directly by reducing the lattice L using LLL [HPS08] as long as $p^a \ll p^{1-\frac{1}{n}}$, *i.e.* $n > \frac{1}{1-a}$.

The attack can also be carried out when $g > 1$: in that case, LLL will recover $\pm 1/g \cdot \boldsymbol{\epsilon}$, so exhaustive search on the few possible values of g is enough. However, the probability that $g > 1$ is so small makes this refinement unnecessary.

Table 5.8: Timings for a SAGE implementation on a 2.27 GHz Intel Core i3 CPU core.

Size of p (modulus size)	256 bits
Number of errors (n)	9
Error size (percentage of the modulus size)	224 bits (87.5%)
Success probability	99.8%
CPU time	3 ms

Experimental results. To evaluate the attack, we implemented it in SAGE [Ste12] (without treating the case $g > 1$) and observed the results given in Table 5.8. The failure rate of $\approx 0.2\%$ corresponds to the cases when $g > 1$, and is consistent with $1/\zeta(9) \approx 0.998$.

5.6.3.2 Several Faulty Results and no Correct Result

Now, assume that the attacker has no access to the correctly converted affine coordinates and that all he gets are the values

$$x_i = x + xZ^2\epsilon_i$$
$$y_i = y + yZ^2\epsilon_i$$

for $i = 0, \ldots, n$. The attack of the previous paragraph extends to this setting. Indeed, Z, x and y can be recovered as follows.

Step 1: Recovering xZ^2, yZ^2 and $\epsilon_i - \epsilon_0$. We have

$$x_i - x_0 = xZ^2 \cdot (\epsilon_i - \epsilon_0)\,.$$

The $(\epsilon_i - \epsilon_0)$ are small. The attacker can hence reuse LLL as in Section 5.6.3.1 with the values $x_i - x_0$ and $\boldsymbol{\epsilon} = (\epsilon_1 - \epsilon_0, \ldots, \epsilon_n - \epsilon_0)$. This will recover xZ^2 and $\boldsymbol{\epsilon}$. The same can be done with y to recover yZ^2.

Step 2: Recovering Z. Let $u = xZ^2$ and $v = yZ^2$ and substitute these values into equation (3.4):

$$\left(\frac{v}{Z^2}\right)^2 = \left(\frac{u}{Z^2}\right)^3 + a\frac{u}{Z^2} + b \quad \text{hence} \quad v^2Z^2 = u^3 + auZ^4 + bZ^6\,.$$

Z^2 can thus be recovered by solving a cubic algebraic equation.

Step 3: Recovering x and y. From $x_1 - x_0 = xZ^2 \cdot (\epsilon_1 - \epsilon_0)$, compute:

$$x = \frac{Z^2 \cdot (\epsilon_1 - \epsilon_0)}{x_1 - x_0}\,.$$

Indeed, Z^2, $(\epsilon_1 - \epsilon_0)$ and $(x_1 - x_0)$ are all known to the attacker. The same holds for y.

In Summary. Several faulty conversions allow to recover the missing Z coordinate.

This attack should not jeopardize standard ECDSA signatures, as a fresh random scalar k is generated during each subsequent run. A schematic presentation of the standard ECDSA is given in Section 3.3, while the Sign and Verify algorithms are given in more detail in Algorithms 29 and 30. However, deterministic signature scheme such as [MNPV98] are vulnerable to this attack. This signature scheme is recalled in appendix B.

Since several faulty results with the same Z coordinate are necessary, any randomization used against side channel attacks, *e.g.* scalar randomization [Cor99, §5.1], input blinding [Cor99, §5.2], random projective coordinates [Cor99, §5.3] or random curve isomorphism [JT01] will thwart this attack.

5.6.4 Two Faults and a Correct Result

As we have just seen, a correct conversion and two faulty conversions yield the values: $Z^2\epsilon_1$ and $Z^2\epsilon_2$ and hence, by modular division $\alpha = \epsilon_1\epsilon_2^{-1}$. Theorem 5.13 (see [FSW02]) guarantees that ϵ_1 and ϵ_2 can be efficiently recovered from α if each ϵ_i is smaller than the square root of p. This problem is known as the *Rational Number Reconstruction* [PW04, WP03] and is typically solved using Gau' algorithm for finding the shortest vector in a bidimensional lattice [Val91].

Theorem 5.13 *Let $\epsilon_1, \epsilon_2 \in \mathbb{Z}$ such that $-A \leq \epsilon_1 \leq A$ and $0 < \epsilon_2 \leq B$. Let $p > 2AB$ be a prime and $\alpha = \epsilon_1\epsilon_2^{-1} \bmod p$. Then ϵ_1, ϵ_2 can be recovered from A, B, α, p in polynomial time.*

Assume that the ϵ_i are smaller than $\sqrt{p}$. Taking $A = B = \lfloor\sqrt{p}\rfloor$, we get $2AB < p$. Moreover, $0 \leq \epsilon_1 \leq A$ and $0 < \epsilon_2 \leq B$. Thus the attacker can recover ϵ_1 and ϵ_2 from α in polynomial time. Note that this attack is a special case of Section 5.6.3.1.

If the ϵ_i are shifted to the left by an arbitrary number of bit positions, this does not change anything as these powers of two will divide out.

The attack is also feasible in the more general unbalanced case when

$$\epsilon_1\epsilon_2 \leq p/4. \tag{5.24}$$

In contrast to the case where the ϵ_i are bound individually (*i.e.* $0 \leq \epsilon_1 \leq A$ and $0 < \epsilon_2 \leq B$) we do not have a fixed bound for ϵ_1 and ϵ_2 anymore; equation (5.24) only provides a bound for the *product* $\epsilon_1\epsilon_2$. Equation (5.24) implies that there exists $1 \leq i \leq \lfloor\log_2 p\rfloor$ such that $0 \leq \epsilon_1 \leq 2^i$ and $0 < \epsilon_2 \leq p/2^{i+1}$. Then using Theorem 5.13 again, the attacker can recover the pair (ϵ_1, ϵ_2), and hence Z. In principle, there could be several candidate solutions depending on the choice of i, making it necessary to consider many possible values of Z. In practice, however, multiple solutions seem to occur with negligible probability when p is large enough.

In Summary. With this attack, the missing Z coordinate can be recovered with two faulty results and one correct result. Faults can have different sizes but the sum of these sizes must not exceed the size of p.

Again, this attack does not threaten standard ECDSA or randomized implementations as it requires two faulty conversions of the same point. It is however still applicable to [MNPV98].

5.6.5 Known or Guessable Faults

If ϵ is known or successfully guessed, then one faulty point $(\tilde{x} = x + xZ^2\epsilon, \tilde{y} = y + yZ^2\epsilon)$ and the correct point (x, y) suffice to recover Z.

5.6.5.1 Attacking ECDSA

Widely adopted by standardisation organism such as ANSI, IEEE and NIST, ECDSA is the elliptic curve version of DSA proposed by Vanstone in 1992 [Van92].

The hardness assumption underlying ECDSA is the intractability of the ECDLP (see Section 2.1.2).

The Elliptic Curve Digital Signature Algorithm (ECDSA) [JM99] uses the following curve parameters:

– E, an elliptic curve over some prime base field $\mathbb{F}_p$
– G, a generator of a subgroup of E of order n

A private key is an integer d randomly chosen in $[1, n-1]$. The corresponding public key is $P = [d]G$.

Algorithm 29: Sign

Input: Private key d, hashed and padded message m

Output: Signature (r, s)

1 $k \xleftarrow{\$} [1, n-1]$
2 $Q \leftarrow [k]G$
3 $r \leftarrow x_Q \bmod n$
4 **if** $r = 0$ **then**
5 | **go to** line 1
6 **end if**
7 $i \leftarrow k^{-1} \bmod n$
8 $s \leftarrow i(dr + m) \bmod n$
9 **if** $s = 0$ **then**
10 | **go to** line 1
11 **end if**
12 **return** (r, s)

Algorithm 30: Verify

Input: Public key P, hashed and padded message m, signature (r, s)

Output: `True` or `False`

1 $w \leftarrow s^{-1} \bmod n$
2 $u_1 \leftarrow w \cdot m \bmod n$
3 $u_2 \leftarrow w \cdot r \bmod n$
4 $Q \leftarrow [u_1]G + [u_2]P$
5 $v \leftarrow x_Q \bmod n$
6 **if** $v = r$ **then**
7 | **return** `True`
8 **end if**
9 **else**
10 | **return** `False`
11 **end if**

We suppose that, during Sign, a fault corrupted the conversion of Q and thus has damaged x_Q. The corresponding erroneous value is denoted $\tilde{x_Q}$. $\tilde{x_Q}$ and x_Q verify equation (5.20). The erroneous signature $(\tilde{r}, \tilde{s})$ satisfies:

$$\begin{aligned} \tilde{r} &= \tilde{x}_Q \mod n \\ \tilde{s} &= i(d \cdot \tilde{r} + m) \mod n \end{aligned}$$

From $(\tilde{r}, \tilde{s})$, the attacker can compute:

$$\begin{aligned}
\tilde{w} &= \tilde{s}^{-1} \mod n \\
\tilde{u}_1 &= \tilde{w} \cdot m \mod n \\
\tilde{u}_2 &= \tilde{w} \cdot \tilde{r} \mod n \\
\tilde{Q} &= [\tilde{u}_1]G + [\tilde{u}_2]P = \left[\tfrac{km}{d\tilde{r}+m}\right] G + \left[\tfrac{k\tilde{r}}{d\tilde{r}+m}\right] P = \left[\tfrac{km}{d\tilde{r}+m}\right] G + \left[\tfrac{dk\tilde{r}}{d\tilde{r}+m}\right] G \\
&= \left[k \cdot \tfrac{d\tilde{r}+m}{d\tilde{r}+m}\right] G = [k]G
\end{aligned}$$

The value $\tilde{Q}$ is hence the correct value of $[k]G$. Thus, if the attacker can guess ϵ, then the attack of [NSS04] becomes possible and some bits of k are disclosed. The attacker can repeat this scenario and obtain several signatures for each of which a few bits of k_i are known. This is precisely the scenario considered in [HGS01] allowing to recover the private key d.

In Summary. This attack requires only one result wrongly converted ECDSA signature under a known fault to recover Z. When repeated, this attack permits to recover the signer's private key.

As opposed to the previous attacks, scalar randomization [Cor99, §5.1] and random projective coordinates [Cor99, §5.3] do not seem to thwart this attack.

5.6.6 Final Observations

Synthesis of the Feasibility of the Attacks. From the analysis of Sections 5.6.1.5, 5.6.1.6 and the analysis of the three different fault attacks, we give a summary of the feasibility of each attack depending on the ECSM and countermeasures used. ✓ indicates that the ECSM or the countermeasure thwarts the attack and × indicates that it does not.

The injection of an error ϵ before the squaring of r *i.e.* right before the operation r^2) will yield

$$\frac{x'}{x} - 1 = U(2 + U)$$

and

$$\frac{y'}{y} - 1 = U(U^2 + 3U + 3),$$

where $U = \epsilon Z$.

Alternatively, the injection of an error ϵ into s just after the operation $x = X \cdot s$ yields a correct x and a faulty y'. Here $\frac{y'}{y} - 1 = \epsilon Z^2$ where y can be derived from x using the curve's equation.

Table 5.9: Synthesis of the attacks.

		Fault Model		
		Large Faults Section 5.6.3	Two Faults Section 5.6.4	Known Fault Section 5.6.5
ECSM	Double-and-Add	×	×	×
	Double-and-Add always	×	×	×
	Signed Digit method	×	×	×
	Sliding Window	×	×	×
	Signed Sliding Window	×	×	×
	Montgomery Ladder	✓	✓	✓
	co-Z Montgomery Ladder	×	×	×
Countermeasures	Random Projective Coordinates *before* ECSM	✓	✓	×
	Random Projective Coordinates *after* ECSM	✓	✓	✓
	Random Curve Isomorphism	✓	✓	×
	Scalar Randomization	✓	✓	×
	Point Blinding	✓	✓	✓
	Point Verification *before* conversion	×	×	×
	Point Verification *after* conversion	✓	✓	✓

CHAPTER 6

CONCLUSION AND FURTHER DEVELOPMENT

This thesis addressed various topics in cryptology. Starting from protocol design, passing through algorithmic improvements and getting to attacks, both authentication and encryption were considered. As an extension, various cryptographic techniques were successfully applied to error correcting codes.

A brief historical overview of cryptography is given in the introduction. We further gave technical details of both symmetric and asymmetric key cryptography, defining the corresponding security notions. We list our publications, pre-prints and our submission to the cryptographic competition CAESAR, which is currently a second-round finalist.

Theoretical foundations underlying the discussed results were presented.

We briefly overviewed basic concepts of hash functions, MACs and authenticated encryption: both the generic composition paradigm and the one-pass solution were presented. An overview of digital signatures was further given. The basics of elliptic curves were recalled and their importance in cryptography was discussed.

The thesis' core results are the following: we replace the individual signatures of two parties by one co-signature to circumvent the restrictive definitions of fairness and viability that do not apply to co-signatures. We then introduce a new notion, called *legal fairness*, defined as the requirement that any transferable proof of involvement of one party with a message also ties the other.

We implemented legal fairness using Schnorr signatures, and proved the construction's security. The proposed paradigm is efficient, compact, fully distributed, fully dynamic, and provably secure in the Random Oracle Model. The protocol is also flexible (meaning that any arbitrary subset of users can co-sign a message) but extending legal fairness for more than two co-signers remains an open problem.

A new nonce-based authenticated encryption scheme (AEAD), called Offset Merkle-Damgard (OMD), offering several attractive features was presented. Unlike the mainstream schemes which are either block-cipher-based or permutation-based schemes, OMD was designed as a mode of operation for a compression function. The design rationale included attaining a provable security relying only on a single well-established standard assumption on the underlying primitive (*i.e.* the PRF assumption on the keyed compression function). Further design guidelines were constructing the scheme based on a simple (as possible) structure and proposing a different approach in view of scheme's variation.

A distributed Fiat-Shamir authentication protocol is presented next. This protocol enables network authentication using very few communication rounds, thereby alleviating the burden of resource-limited devices such as wireless sensors and other IoT nodes. Instead of performing one-to-one authentication to check the integrity of the network seen a whole (a single entity), our protocol gives a collective proof of integrity for the whole network at once.

Lightweight cryptography solutions were described, initially for RFID tags. Possible risks were examined and different measures for improving their level of security and privacy were discussed.

A method allowing to double the speed of Barrett's algorithm by using specific composite moduli was presented. The technique is particularly useful for lightweight devices where such an optimization can make a difference in terms of power consumption, cost and processing time. The generation of composite moduli with a predetermined portion is a well-known technique and the use of such moduli is considered, *in statu scientiæ*, as safe as using randomly generated composite moduli. We also suggested a modified DSA prime generation process leveraging the idea.

We described a new backtracking-based multiplication algorithm, especially tailored for lightweight microprocessors when one of the operands is known in advance.

A polynomial extension of Barrett's modular reduction algorithm was presented. Optimisations were discussed and applied to error correction. The goal was the efficient implementation of an error-correction code (BCH), suitable for constrained memory specifications. The implementation of the proposed polynomial variant of Barrett's algorithm confirms that the new algorithm allows to reduce hardware complexity and hence save resources. Synthesis results for different ECC circuits reveal various trade-off possibilities and provide guidelines for choosing a particular hardware architecture depending on the application's requirements.

We proposed a new building block denoted *Pace Regulator* and applied it for streamlining the pace of random bits outputt by a von Neumann extractor.

We presented a new error-correcting code based on number theory, as well as a form of message size improvement based on the hybrid use of two ECCs. This construction is inspired by the Naccache-Stern (NS) cryptosystem [NS97, CMNS08].

If the output of scalar multiplication $[k]P$ on an ECC is given in Jacobian coordinates (X, Y, Z), information on k can be recovered [NSS04]. Building upon [NSS04], we proposed a new fault attack on ECC implementations. The attack consists in injecting a fault during the projective-to-affine conversion process so the erroneous results reveals information about Z. Several faulty results allow recovering Z and hence, expose a signer to the attack described in [NSS04]. The attack comes in several variants: if the error is known or guessed, the attack requires only one faulty point. If the error is unknown then several faults are necessary. In ECDSA the ability to inject known errors in conjunction with our attacks allows to recover the signer's secret key.

As further development, we proposed new authenticated encryption ideas in Section 4.3.11.2: obtaining the tag as a by-product of encryption, allowing the leakage of certain bits to achieve authentication.

Another research direction we consider is instantiating OMD with a compression function more suitable for hardware implementations in terms of efficiency. When analyzing the implications of such a construction resistance to various types of attacks (even side channel attacks) should be taken into account.

Based on our *Pace Regulator* presented in Section 5.5, we propose practical implementations for cryptographic hardware. Furthermore, we recommend analyzing this construction's benefits seen as a countermeasure tool, *e.g.* masking the properties of a True Random Number Generator implemented in cryptographic hardware.

BIBLIOGRAPHY

[AABN02] M. Abdalla, J. An, M. Bellare, and C. Namprempre. From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security. In *Advances in Cryptology - EUROCRYPT'98*, volume 2332 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2002. 60

[AD97] M. A. Ajtai and C. Dwork. A Public-key Cryptosystem with Worst-Case/Average-Case Equivalence. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC'97, pages 284–293. ACM, 1997. 30

[ADN+10] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria. When Clocks Fail: On Critical Paths and Clock Faults. In *Smart Card Research and Advanced Application*, volume 6035 of *Lecture Notes in Computer Science*. Springer, 2010. 146

[Ajt96] M. A. Ajtai. Generating Hard Instances of Lattice Problems (Extended Abstract). In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, STOC'96, pages 99–108. ACM, 1996. 30

[AK98] R. J. Anderson and M. G. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *Proceedings of the 5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 1998. 145, 147

[And49] W. André. Numbers of Solutions of Equations in Finite Fields. *Bulletin of the American Mathematical Society*, 55(5):497–508, May 1949. 55

[And14] In *Advances in Cryptology – ASIACRYPT'14*. 2014. 48, 94

[AOS02] M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n Signatures from a Variety of Keys. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432. Springer, December 2002. 59, 62

[APW09] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext Recovery Attacks Against SSH. In *Proceedings of the 30th IEEE Symposium on Security and Privacy - SP'09*, pages 16–26. IEEE Computer Society, 2009. 42, 47

[AR05] D. Anshul and S. Roy. A ZKP-Based Identification Scheme for Base Nodes in Wireless Sensor Networks. In *Proceedings of the 20th ACM Symposium on Applied Computing - SAC'05*, pages 319–323. ACM, 2005. 72

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In *ACM CCS 97: 4th Conference on Computer and Communications Security*, pages 7–17. ACM Press, April 1997. 59

[Avi61] A. Avižienis. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers*, (3):389–400, 1961. 132

[Bar87] P. Barrett. Implementing the Rivest, Shamir and Adleman Public-Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology - CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1987. 104, 113, 132

[BB12] W. C. Barker and E. Barber. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. NIST Special Publication 800-67 Revision 1, January 2012. 10, 20

[BBBV97] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing*, 26(5):1510–1523, October 1997. 31

[BBC+08] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. SOSEMANUK: A Fast Software-Oriented Stream Cipher. *CoRR*, abs/0810.1858, 2008. 10, 20

[BBKN12] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, November 2012. 147

[BBPS11] A. Barenghi, G. Bertoni, A. Palomba, and R. Susella. A Novel Fault Attack Against ECDSA. In *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust - HOST'11*, pages 161–166, 2011. 145

[BCG+15a] E. Brier, J. S. Coron, R. Géraud, D. Maimuţ, and D. Naccache. A Number-Theoretic Error-Correcting Code. *CoRR*, abs/1509.00378:302, 2015. 35

[BCG+15b] E. Brier, J. S. Coron, R. Géraud, D. Maimuţ, and D. Naccache. A Number-Theoretic Error-Correcting Code. In *SECITC'15*, Lecture Notes in Computer Science. Springer, to appear in 2015. 35

[BCN+04] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *IACR Cryptology ePrint Archive*, 2004:100, 2004. 145, 146

[BD08] S. Babbage and M. Dodd. The MICKEY Stream Ciphers. In *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008. 10, 20

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and Ph. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Proceedings of the 38th International IEEE Symposium on Foundations of Computer Science - FOCS'97*, pages 394–403, 1997. 25, 26, 28, 46

[BDL01] D. Boneh, R. A. Demillo, and R. J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14:101–119, 2001. 145

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and Ph. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Advances in Cryptology - CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer-Verlag, 1998. 28, 29

[BDPS12] A. Boldyreva, Jean P. Degabriele, K. G. Paterson, and M. Stam. Security of Symmetric Encryption in the Presence of Ciphertext Fragmentation. In *Advances in Cryptology - EUROCRYPT'12*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699. Springer, 2012. 45

[BDPVA09] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak Specifications, 2009. 39

[Bel06] M. Bellare. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. *IACR Cryptology ePrint Archive*, 2006:43, 2006. 80

[Bel11] S. M. Bellovin. Frank Miller: Inventor of the One-Time Pad. *Cryptologia*, 35(3):203–222, 2011. 20

[Ber] D. J. Bernstein. Cryptographic Competitions: CAESAR. `http://competitions.cr.yp.to`. 33

[Ber86] R. Bernstein. Multiplication by Integer Constants. *Software - Practice and Experience*, 16(7):641–652, 1986. 107, 118, 132, 133

[Ber05] D. J. Bernstein. Cache-Timing Attacks on AES, 2005. 81, 94

[Ber08] D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008. 10, 20

[Ber09] Lattice-based Cryptography. In *Post-Quantum Cryptography*, pages 147–191. Springer, 2009. 30

[BF01] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. 2139:213–229, 2001. 56

[BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology — EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003. 72

[BGM04] M. Bellare, O. Goldreich, and A. Mityagin. The Power of Verification Queries in Message Authentication and Authenticated Encryption. *IACR Cryptology ePrint Archive*, 2004:309, 2004. 89, 92

[BGM06] C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In *Fast Software Encryption - FSE'06*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006. 20

[BGR95] M. Bellare, R. Guérin, and Ph. Rogaway. XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In *Advances in Cryptology - CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 1995. 83

[BGR98] M. Bellare, J. A. Garay, and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *Advances in Cryptology - EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998. 78

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes. In *IEEE International Conference on Communications - ICC'93*, volume 2, pages 1064–1070, May 1993. 124

[BGV94] A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of Three Modular Reduction Functions. In *Advances in Cryptology - EUROCRYPT'93*, volume 773 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 1994. 104, 113

[BHH+15] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *Lecture Notes in Computer Science*, pages 368–397. Springer, 2015. 31

[BKN04] M. Bellare, T. Kohno, and C. Namprempre. Breaking and Provably Repairing the SSH Authenticated Encryption Scheme: A Case Study of the Encode-then-Encrypt-and-MAC Paradigm. *ACM Transactions on Information and System Security - TISSEC'04*, 7(2):206–241, 2004. 42, 45, 47

[BKW03] A. Blum, A. Kalai, and H. Wasserman. Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model. *Journal of the ACM*, 50(4):506–519, 2003. 101

[Ble96] D. Bleichenbacher. Generating EIGamal Signatures Without Knowing the Secret Key. In *Advances in Cryptology — EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1996. 51

[BLP08] D. J. Bernstein, T. Lange, and C. Peters. Attacking and Defending the McEliece Cryptosystem. In *Post-Quantum Cryptography*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin Heidelberg, 2008. 31

[BMM00] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Advances in Cryptology - CRYPTO'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000. 145

[BMvT06] E. Berlekamp, R. McEliece, and H. van Tilborg. On the Inherent Intractability of Certain Coding Problems (Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, September 2006. 31

[BN00] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology - ASIACRYPT'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000. 46, 47, 80

[BN08] M. Bellare and C. Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. *Journal of Cryptology*, 21(4):469–491, 2008. 12, 32, 44, 45

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *Proceedings of the 3rd International Algorithmic Number Theory Symposium on - ANTS'98*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998. 55

[Bos11] A. Bosselaers. RIPEMD Family. In *Encyclopedia of Cryptography and Security*, pages 1050–1053. Springer, 2nd edition, 2011. 39

[BOSW88] M. Ben-Or, Goldwasser. S., and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, May 1988. 59

[BR93] M. Bellare and Ph. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security - CCS'93*, pages 62–73. Fairfax, 1993. 53

[BR94] M. Bellare and Ph. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology - CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1994. 53

[BR00] M. Bellare and Ph. Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In *Advances in Cryptology - ASIACRYPT'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000. 80

[Bri83] E. F. Brickell. A Fast Modular Multiplication Algorithm with Applications to Two Key Cryptography. In *Advances in Cryptology - CRYPTO'82*, volume 330, pages 51–60. Plenum, 1983. 104, 113, 118, 121

[BRW03] M. Bellare, P. Rogaway, and D. Wagner. EAX: A Conventional Authenticated-Encryption Mode, 2003. 43

[BS91] E. Biham and A. Shamir. Differential Cryptanalysis of DES-Like Cryptosystems. In *Advances in Cryptology - CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer-Verlag, 1991. 10, 20

[BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology - CRYPTO'97*, Lecture Notes in Computer Science, pages 513–525. Springer-Verlag, 1997. 145

[BSS99] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999. 149

[BV93] E. Bernstein and U. Vazirani. Quantum Complexity Theory. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 11–20. ACM, 1993. 30

[BVP+03] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius. Rabbit: A New High-Performance Stream Cipher. In *Fast Software Encryption*, volume 2887 of *Lecture Notes in Computer Science*, pages 307–329. Springer-Verlag, 2003. 10, 20

[BW00] B. Baum-Waidner and M. Waidner. Round-Optimal and Abuse Free Optimistic Multi-party Contract Signing. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 524–535. Springer, 2000. 59, 62

[CC00] C. Cachin and J. Camenisch. Optimistic Fair Secure Computation. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer, August 2000. 59

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19. ACM Press, May 1988. 59

[Cer] Certivox. The MIRACL big number library. `www.certivox.com/miracl`. 133

[CFA+12] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2nd edition, 2012. 55, 56

[Cha82] D. Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology - CRYPTO'82*, pages 199–203. Plenum Press, 1982. 50

[CHVV03] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *Advances in Cryptology - CRYPTO'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599. Springer, 2003. 80

[CJ05] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Journal of Designs, Codes and Cryptography*, 36(1):33–43, 2005. 145

[CKP04] L. Chen, C. Kudla, and K. G. Paterson. Concurrent Signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 287–305. Springer, May 2004. 59, 61, 62, 63

[Cle86] R. Cleve. Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract). In J. Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369. ACM, 1986. 59

[CMN+14] S. Cogliani, D.-Ş. Maimuţ, D. Naccache, R. Portella do Canto, R. Reyhanitabar, S. Vaudenay, and D. Vizár. OMD: A Compression Function Mode of Operation for Authenticated Encryption. In *Selected Areas in Cryptography - SAC'14*, volume 8781 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2014. 33

[CMNS08] B. Chevallier-Mames, D. Naccache, and J. Stern. Linear Bandwidth Naccache-Stern Encryption. In *Proceedings of the 6th International Conference on Security and Cryptography for Networks - SCN '08*, volume 5229 of *Lecture Notes in Computer Science*, pages 327–339. Springer, 2008. 13, 32, 124, 129, 157

[CND+06] J.-S. Coron, D. Naccache, Y. Desmedt, A. Odlyzko, and J. P. Stern. Index Calculation Attacks on RSA Signature and Encryption. *Designs, Codes and Cryptography*, 38(1):41–53, 2006. 51

[CNS99] J.-S. Coron, D. Naccache, and J. P. Stern. On the Security of RSA Padding. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 1999. 51

[Cona] Wikipedia Contributors. Cold Boot Attack. In *Wikipedia, The Free Encyclopedia*. 146

[Conb] Wikipedia Contributors. ESTREAM. In *Wikipedia, The Free Encyclopedia*. 20

[Coo66] S. A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, 1966. 132

[Cor99] J.-S. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems - CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999. 149, 150, 152, 154

[CS84] P. R. Cappello and K. Steiglitz. Some Complexity Issues in Digital Signal Processing. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 32(5):1037–1041, 1984. 132

[CS08] D. Chakraborty and P. Sarkar. A General Construction of Tweakable Block Ciphers and Different Modes of Operations. *IEEE Transactions on Information Theory*, 54(5):1991–2006, 2008. 84, 93

[CSRL01] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. 74

[DCDK09] C. De Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'09*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009. 99

[DCP06] C. De Canniere and B. Preneel. TRIVIUM Specifications. *eSTREAM, ECRYPT Stream Cipher Project*, 2006. 10, 20

[DD08] M. Douguet and V. Dupaquis. Modular Reduction Using a Special Form of the Modulus. In *U.S. Patent Application 12/033,512*, Atmel Corporation, 2008. filed February 19, 2008. 104

[DDN00] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *SIAM Journal on Computing*, pages 542–552. Society for Industrial and Applied Mathematics, 2000. 28

[DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. 11, 22, 50, 132

[Dif88] W. Diffie. Innovations in Internetworking. In *The First Ten Years of Public-key Cryptography*, pages 510–527. Artech House, 1988. 11, 22

[DM94] A. G. Dempster and M. D. Macleod. Constant Integer Multiplication using Minimum Adders. *IEE Proceedings - Circuits, Devices and Systems*, 141(5):407–413, 1994. 132

[DM95] A. G. Dempster and M. D. Macleod. Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 42(9):569–577, 1995. 132

[DR98] J. Daemen and V. Rijmen. AES Proposal: Rijndael, 1998. 10, 20

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002. 31

[DR11] T. Duong and J. Rizzo. BEAST: Surprising Crypto Attack Against HTTPS. In *Blog*, September 2011. 42, 45, 47

[Dus99] P. Dusart. The k-th Prime is Greater than $k(\ln k + \ln \ln k - 1)$ for $k \geq 2$. *Mathematics of Computation*, pages 411–415, 1999. 130

[Dwo01] M. Dworkin. Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A, December 2001. 11, 20

[Dwo10] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. NIST Special Publication 800-38E, January 2010. 11, 20

[EG84] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984. 11, 22, 50, 51, 60, 64, 132

[Eli55] P. Elias. Coding for Noisy Channels. In *IRE Wescon Convention Record*, pages 37–46, 1955. 124

[FD86] H. J. Fell and W. Diffie. Analysis of a Public Key Approach Based on Polynomial Substitution. In *Advances in Cryptology - CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 340–349. Springer-Verlag, 1986. 31

[Fey82] R. P. Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21(6/7), 1982. 30

[FFL12] E. Fleischmann, C. Forler, and S. Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012. 46, 80

[FFS87] U. Feige, A. Fiat, and A. Shamir. Zero Knowledge Proofs of Identity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, USA*, pages 210–217, 1987. 132

[FFS88] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, 1(2):77–94, 1988. 60, 72, 132

[FGM+15a] H. Ferradi, R. Géraud, D. Maimuţ, D. Naccache, and A. de Wargny. Regulating the Pace of von Neumann Correctors. *IACR Cryptology ePrint Archive*, 2015:849, 2015. 36

[FGM+15b] H. Ferradi, R. Géraud, D. Maimuţ, D. Naccache, and H. Zhou. Backtracking-Assisted Multiplication. *IACR Cryptology ePrint Archive*, 2015:787, 2015. 35

[FGV11] J. Fan, B. Gierlichs, and F. Vercauteren. To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order. In *Proceedings of the 13th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'11*, volume 6917 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2011. 145

[fip85] *Computer Data Authentication*. National Institute of Standards and Technology, May 1985. Note: Federal Information Processing Standard 113. 40

[fip00] *Digital Signature Standard (DSS)*. National Institute of Standards and Technology, 2000. Note: Federal Information Processing Standard 186-2. 104

[FIP12] Secure Hash Standard (SHS) . NIST FIPS PUB 180-4, March 2012. 39, 80, 85, 87, 177, 178, 180

[fip13] *Digital signature standard (DSS)*. National Institute of Standards and Technology, 2013. Note: Federal Information Processing Standard 186-4. 104, 110

[FKL+05] S. E. Frankel, K. Kent, R. Lewkowski, A. D. Orebaugh, R. W. Ritchey, and S. R. Sharma. Guide to IPsec VPNs. Technical Report 800-38A, 2005. 42

[FLRV08] P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault Attack on Elliptic Curve Montgomery Ladder Implementation. In *Proceedings of Fault Diagnosis and Tolerance in Cryptography - FDTC'08*, pages 92–98. IEEE Computer Society, 2008. 145

[Fou98] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., 1998. 10, 20

[FPS12] S. Faust, K. Pietrzak, and J. Schipper. Practical Leakage-Resilient Symmetric Cryptography. In *Proceedings of the 14th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'12*, volume 7428 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2012. 46

[FS87] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987. 50, 51, 72, 73, 75

[FSW02] P.-A. Fouque, J. Stern, and J.-G. Wackers. CryptoComputing with Rationals. In *Proceedings of the 6th International Conference - Financial Cryptography'02*, volume 2357 of *Lecture Notes in Computer Science*, pages 136–146. Springer, 2002. 125, 152

[Für09] M. Fürer. Faster Integer Multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. 132

[GAL91] S. Goldwasser and L. A. A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, August 1991. 59

[GD01] V. D. Gligor and P. Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In *Fast Software Encryption - FSE'01*, volume 2355 of *Lecture Notes in Computer Science*, pages 92–108, 2001. 12, 32

[GDMPY06] J. A. Garay, P. D. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource Fairness and Composability of Cryptographic Protocols. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 404–428. Springer, March 2006. 59

[GHKL08] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 413–422. ACM Press, May 2008. 59

[GJDM99] J. A. Garay, M. Jakobsson, and P. D. D. MacKenzie. Abuse-Free Optimistic Contract Signing. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer, August 1999. 59, 62

[GJM10] R. R. Goundar, M. Joye, and A. Miyaji. Co-z Addition Formulæand Binary Ladders on Elliptic Curves - (Extended Abstract). In *Proceedings of the 12th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'10*, volume 6225 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2010. 149, 181, 182

[GK04] C. Giraud and E. W. Knudsen. Fault Attacks on Signature Schemes. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy - ACISP'04*, volume 3108 of *Lecture Notes in Computer Science*, pages 478–491. Springer, 2004. 145

[GKM+13] J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Rational Protocol Design: Cryptography Against Incentive-Driven Adversaries. In *Proceedings of the IEEE 54th Annual Symposium on Foundations of Computer Science - FOCS'13*, pages 648–657. IEEE Computer Society, 2013. 32

[GKT10] C. Giraud, E. W. Knudsen, and M. Tunstall. Improved Fault Analysis of Signature Schemes. In *Proceedings of CARDIS'10*, volume 6035 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2010. 145

[GM82] S. Goldwasser and S. Micali. Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing - STOC'82*, pages 365–377. ACM, 1982. 28

[GM84] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Sciences*, 28(2):270–299, 1984. 25, 28, 45, 53

[GMN15a] R. Géraud, D. Maimuţ, and D. Naccache. Double-Speed Barrett Moduli. *IACR Cryptology ePrint Archive*, 2015:785, 2015. 34

[GMN+15b] R. Géraud, D. Maimuţ, D. Naccache, R. Portella do Canto, and E. Simion. Applying Cryptographic Acceleration Techniques to Error Correction. *IACR Cryptology ePrint Archive*, 2015:886, 2015. 35

[GMN15c] R. Géraud, D. S. Maimuţ, and D. Naccache. Double-Speed Barrett Moduli. In *Kahn Festschrift*, volume 9100 of *Lecture Notes in Computer Science*. Springer, to appear in 2015. 34

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, STOC'85, pages 291–304. ACM, 1985. 50, 72

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM Press, May 1987. 59

[GO03] K. Gaj and A. Orlowski. Facts and Myths of Enigma: Breaking Stereotypes. In *Advances in Cryptology - EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2003. 9, 18

[Gol83] O. Goldreich. A Simple Protocol for Signing Contracts. In D. Chaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983.*, pages 133–136. Plenum Press, New York, 1983. 61

[Gol04] O. Goldreich. Preface. *Journal of Cryptology*, 17(1):1–3, 2004. 59

[Gop81] V. D. Goppa. Codes on Algebraic Curves. *Soviet Math. Doklady*, 24:170–172, 1981. 124

[GPR14] P. Gazi, K. Pietrzak, and M. Rybár. The Exact PRF-Security of NMAC and HMAC. In *Advances in Cryptology - CRYPTO'14*, volume 8616 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2014. 80

[GPS06] M. Girault, G. Poupard, and J. Stern. On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order. *Journal of Cryptology*, 19(4):463–487, 2006. 51, 60, 64

[GQ88] L. C. Guillou and J.-J. Quisquater. A Practical Zero-knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 1988. 72

[Gro96] L. K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing - STOC'96*, pages 212–219. ACM, 1996. 31

[GRS05] H. Gilbert, M. J. B. Robshaw, and H. Sibert. An Active Attack Against HB^+ - A Provably Secure Lightweight Authentication Protocol. *IACR Cryptology ePrint Archive*, 2005:237, 2005. 102

[GRS08] H. Gilbert, M. J. B. Robshaw, and Y. Seurin. $HB^\#$: Increasing the Security and Efficiency of HB+. In *Advances in Cryptology - EUROCRYPT'08*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2008. 101, 102

[GS94] M. Girault and J. Stern. On the Length of Cryptographic Hash-Values Used in Identification Schemes. In *Advances in Cryptology - CRYPTO'94*, pages 202–215, 1994. 73

[Gut96] P. Gutmann. Secure Deletion of Data from Magnetic and Solid-state Memory. In *Proceedings of the 6th Conference on USENIX Security Symposium*, SSYM'96, pages 8–8. USENIX Association, 1996. 146

[Hal02] S. Hallgren. Polynomial-Time Quantum Algorithms for Pell's Equation and the Principal Ideal Problem. In *STOC*, pages 653–658. ACM, 2002. 30

[Ham50] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29(2):147–160, 1950. 124

[Har08] D. Harkind. Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES). IETF RFC 5297 (Informational), October 2008. 43

[Has35] H. Hasse. *Zur Theorie der abstrakten elliptischen Funktionenkörper*. Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen. Mathematisch-physikalische Klasse. Fachgruppe 1. Vandenhoeck & Ruprecht, 1935. 55

[HB01] N. J. Hopper and M. Blum. Secure Human Identification Protocols. In *Advances in Cryptology — ASIACRYPT'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001. 101

[HGS01] N. Howgrave-Graham and N. P. Smart. Lattice Attacks on Digital Signature Schemes. 23(3):283–290, 2001. 154

[HJM07] M. Hell, T. Johansson, and W. Meier. Grain: A Stream Cipher for Constrained Environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007. 10, 20

[HPM94] P. Horster, H. Petersen, and M. Michels. Meta-El-Gamal Signature Schemes. In *ACM CCS 94: 2nd Conference on Computer and Communications Security*, pages 96–107. ACM Press, 1994. 60, 64

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In *Proceedings of the 3rd International Algorithmic Number Theory Symposium on - ANTS'98*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998. 30

[HPS08] J. Hoffstein, J. Pipher, and J. H. Silverman. *An Introduction to Mathematical Cryptography*. Undergraduate Texts in Mathematics. Springer, 2008. 151

[HVDHL14] D. Harvey, J. Van Der Hoeven, and G. Lecerf. Even Faster Integer Multiplication. *CoRR*, abs/1407.3360, 2014. 132

[IEE99] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, June 1999. 42

[Int13] Intel® SHA Extensions, July 2013. 80

[IOM12] T. Iwata, K. Ohashi, and K. Minematsu. Breaking and Repairing GCM Security Proofs. In *Advances in Cryptology - CRYPTO'12*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012. 47, 80

[iso12] ISO/IEC 29192-1:2012: Information technology - Security techniques - Lightweight cryptography - Part 1: General, 2012. 99

[JM99] D. Johnson and A. Menezes. The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, Department of C&O, University of Waterloo, 1999. 56, 153

[JMV02] A. Joux, G. Martinet, and F. Valette. Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provably Secure Encryption Models: CBC, GEM, IACBC. In *Advances in Cryptology - CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2002. 45

[Jou] A. Joux. 137

[Jou00] A. Joux. A One Round Protocol for Tripartite Diffie–Hellman. In *Proceedings of the 4th International Algorithmic Number Theory Symposium on - ANTS'00*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000. 56

[Joy08] M. Joye. RSA Moduli with a Predetermined Portion: Techniques and Applications. In *Information Security Practice and Experience*, volume 4991 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2008. 104, 106, 118

[JSI96a] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and Their Applications. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer, 1996. 50

[JSI96b] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and Their Applications. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer, May 1996. 59

[JT01] M. Joye and C. Tymen. Protections Against Differential Analysis for Elliptic Curve Cryptography. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems - CHES'01*, volume 2162 of *Lecture Notes in Computer Science*, pages 377–390. Springer, 2001. 149, 152

[JT12] M. Joye and M. Tunstall. *Fault Analysis in Cryptography*. Springer, 2012. 147

[Jut01] C. S. Jutla. Encryption Modes with Almost Free Message Integrity. In *Advances in Cryptology - EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001. 12, 32

[JW05] A. Juels and S. A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology - CRYPTO'05*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005. 101, 102

[JY02] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'02*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002. 149

[Kah96] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, rev. sub. edition, December 1996. 7, 9, 16, 17, 18

[KBV09] M. Knezevic, L. Batina, and I. Verbauwhede. Modular Reduction without Precomputational Phase. In *International Symposium on Circuits and Systems ISCAS*, pages 1389–1392, 2009. 104

[Kho14] K. A. Khoureich. *h*HB: a Harder HB+ Protocol. *IACR Cryptology ePrint Archive*, 2014:562, 2014. 101, 103

[Kno88] H.-J. Knobloch. A Smart Card Implementation of the Fiat-Shamir Identification Scheme. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 87–95. Springer, 1988. 106, 120

[Knu68] D. E. Knuth. The Art of Computer Programming, 1968. 132, 134

[Knu81] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1981. 104, 113, 120

[KO62] A. Karastuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Doklady Akad. Nauk SSSR*, 145(293-294), 1962. 132, 133

[Kob87] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. 55

[Koc96] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. 45

[Koh78] L. Kohnfelder. *Towards a Practical Public-Key Cryptosystem*. PhD thesis, Massachusetts Institute of Technology, May 1978. 50

[KPP+06] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA –A Cost-Optimized Parallel Code Breaker. In *Proceedings of the 8th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'06*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer-Verlag, 2006. 10, 20

[KR11] T. Krovetz and Ph. Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011. 83, 84, 93, 94

[KR14] T. Krovetz and Ph. Rogaway. The OCB Authenticated-Encryption Algorithm, January 2014. 43

[KSS10] J. Katz, J. S. Shin, and A. Smith. Parallel and Concurrent Security of the HB and HB+ Protocols. *Journal of Cryptology*, 23(3):402–421, 2010. 102

[KY01] J. Katz and M. Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In *FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2001. 80

[Lan] J. Landt. The History of RFID. `www.sepaco-tech.com/modules/Manager/Articles/the$%$20history$%$20of$%$20rfid.pdf`. 100

[Len87] H. W. Lenstra. Factoring Integers with Elliptic Curves. *Annals of Mathematics*, 126:649–673, 1987. 55

[Len98] A. K. Lenstra. Generating RSA Moduli with a Predetermined Portion. In *Advances in Cryptology - ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1998. 104, 106, 113

[Lin08] A. Y. Lindell. Legally-enforceable fairness in secure two-party computation. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 121–137. Springer, April 2008. 59, 61

[LMQ+03] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Des. Codes Cryptography*, 28(2):119–134, March 2003. 56

[LVP08] C. Lavault and M. Valencia-Pabon. A Distributed Approximation Algorithm for the Minimum Degree Minimum Weight Spanning Trees. *Journal of Parallel and Distributed Computing*, 68(2):200–208, 2008. 73

[Lys02] A. Lysyanskaya. Unique Signatures and Verifiable Random Functions From the DH-DDH Separation. In *Advances in Cryptology - CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612. Springer, 2002. 56

[McE78] R. J. McEliece. A Public-Key Cryptosystem Based on Algebraic Coding Theory. *DSN progress report*, 42(44):114–116, 1978. 31

[Mei91] G. Meister. On an Implementation of the Mohan-Adiga Algorithm. In *Advances in Cryptology - EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 496–500. Springer, 1991. 106, 120, 121

[Mel07] N. Meloni. New Point Addition Formulae for ECC Applications. In *WAIFI*, volume 4547 of *Lecture Notes in Computer Science*, pages 189–201. Springer, 2007. 181

[Mer] R. Merkle. Establishing Secure Communications between Seperate Secure Sites over Insecure Communication lines. In *The Original CS244 Project Proposal from Fall of 1974*. 11, 22

[Mer79] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, 1979. 31, 39

[MGW03] A. J. Mooij, N. Goga, and J. W. Wesselink. *A Distributed Spanning Tree Algorithm for Topology-Aware Networks*. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 2003. 73, 74

[MH78] R. C. Merkle and M. E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978. 11, 22

[MH81] R. C. Merkle and M. E. Hellman. On the Security of Multiple Encryption. *Communications of the ACM*, 24(7):465–467, 1981. 10, 20

[MI88] T. Matsumoto and H. Imai. Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In *Advances in Cryptology — EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 419–453. Springer-Verlag, 1988. 31

[Mic03] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *22nd ACM Symposium Annual on Principles of Distributed Computing*, pages 12–19. ACM Press, July 2003. 59

[Mil86] V. S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology - CRYPTO 85*, volume 218 of *Lecture Notes in Computer Sciences*, pages 417–426. Springer-Verlag, 1986. 55

[Mis98] J.-F. Misarsky. How (Not) to Design RSA Signature Schemes. In *Public-Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 14–28. Springer-Verlag, 1998. 51

[MLMI12] K. Minematsu, S. Lucks, H. Morita, and T. Iwata. Attacks and Security Proofs of EAX-Prime, 2012. 42

[MMNT13] D. Maimuţ, C. Murdica, D. Naccache, and M. Tibouchi. Fault Attacks on Projective-to-Affine Coordinates Conversion. In *Proceedings of the 4th International Workshop on Constructive Side-Channel Analysis and Secure Design - COSADE 2013*, volume 7864 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2013. 34

[MN96] D. M'Raïhi and D. Naccache. Batch Exponentiation: A Fast DLP-Based Signature Generation Strategy. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security - CCS'96*, pages 58–61. ACM, 1996. 78

[MNPdCS15] D.-Ş. Maimuţ, D. Naccache, R. Portella do Canto, and E. Simion. Applying Cryptographic Acceleration Techniques to Error Correction. In *SECITC'15*, Lecture Notes in Computer Science. Springer, to appear in 2015. 35

[MNPV98] D. M'Raïhi, D. Naccache, D. Pointcheval, and S. Vaudenay. Computational Alternatives to Random Number Generators. In *Selected Areas in Cryptography - SAC'98*, volume 1556 of *Lecture Notes in Computer Science*, pages 72–80. Springer, 1998. 152, 183

[MO12] D. Maimuţ and K. Ouafi. Lightweight Cryptography for RFID Tags. *IEEE Security & Privacy*, 10(2):76–79, 2012. 36

[Mon85] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985. 104, 113, 132

[MOV93] A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993. 56

[MR09] D. Micciancio and O. Regev. Lattice-Based Cryptography. In *Post-Quantum Cryptography*, pages 147–191. Springer, 2009. 30

[MR14] D. Maimuţ and R. Reyhanitabar. Authenticated Encryption: Toward Next-Generation Algorithms. *IEEE Security & Privacy*, 12(2):70–72, 2014. 36

[Mul54] D. E. Muller. Application of Boolean Algebra to Switching Circuit Design and to Error Detection. *IRE Transactions on Information Theory*, (3):6–12, 1954. 124

[MvOV96] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. 20

[MWa] Merriam-Webster. Cryptanalysis. In *Merriam-Webster Dictionary*. Encyclopædia Britannica. 6, 15

[MWb] Merriam-Webster. Cryptology. In *Merriam-Webster Dictionary*. Encyclopædia Britannica. 6, 15

[NC11] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 10th edition, 2011. 30

[NES] NESSIE. List of NESSIE Submissions as Originally Submitted . In *NESSIE Project*. 51

[NIS99] FIPS PUB 46-3, Data Encryption Standard (DES), 1999. U.S.Department of Commerce/-National Institute of Standards and Technology. 20

[NIS01] FIPS PUB 197, Advanced Encryption Standard (AES), 2001. U.S.Department of Commerce/National Institute of Standards and Technology. 10, 20

[NP12] D. Naccache and D. Pointcheval. Autotomic Signatures. In *Quisquater Festschrift*, volume 6805 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2012. 50

[NS97] D. Naccache and J. Stern. A New Public-Key Cryptosystem. In *Advances in Cryptology - EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 27–36. Springer, 1997. 13, 32, 124, 157

[NS98] P. Nguyen and J. Stern. Cryptanalysis of the Ajtai-Dwork Cryptosystem. In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 223–242. Springer Berlin Heidelberg, 1998. 30

[NSS04] D. Naccache, N. P. Smart, and J. Stern. Projective Coordinates Leak. In *Advances in Cryptology - EUROCRYPT'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2004. 13, 32, 33, 97, 145, 147, 148, 149, 150, 154, 157

[OK14] R. O'neil King. National and Civil ID White Paper. Biometrics Research Group, 2014. http://www.scribd.com/doc/231482826/National-and-Civil-ID-Report. 100

[OOV08] K. Ouafi, R. Overbeck, and S. Vaudenay. On the Security of HB$^{\#}$ Against a Man-in-the-Middle Attack. In *Advances in Cryptology - ASIACRYPT'08*, volume 5350 of *Lecture Notes in Computer Science*, pages 108–124, 2008. 102

[oST99] National Institute of Standards and Technology. Recommended Elliptic Curves for Federal Government Use, July 1999. 56

[Pai99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999. 28

[Pin03] Benny Pinkas. Fair secure two-party computation. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer, May 2003. 59

[Poi05] D. Pointcheval. *Advanced Course on Contemporary Cryptology*, chapter Provable Security for Public-Key Schemes, pages 133–189. Advanced Courses CRM Barcelona. Birkhäuser Publishers, Basel, June 2005. 28

[Pos09] A. Y. Poschmann. *Lightweight Cryptography: Cryptographic Engineering for a Pervasive World*. PhD thesis, Ruhr-Universität Bochum, 2009. 98, 99

[pqc] PQCrypto. cordis.europa.eu/project/rcn/194347_en.html. 30

[Pre93] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. 42

[PS96] D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology - EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 1996. 51, 60

[PS00] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, 2000. 51, 53, 60, 67

[PST+02] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security Protocols for Sensor Networks. *Wirel. Netw.*, 8(5):521–534, September 2002. 72

[PW04] V. Y. Pan and X. Wang. On Rational Number Reconstruction and Approximation. *SIAM Journal on Computing*, 33(2):502–503, 2004. 152

[QD90] J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search. New Results and Applications to DES. In *Advances in Cryptology — CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 408–413. Springer, 1990. 39

[Qui92] J. J. Quisquater. Encoding System According to the So-Called RSA Method, by Means of a Microcontroller and Arrangement Implementing this System, 1992. filed November 24, 1992. 104

[Rab79] M. O. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical report, 1979. 39

[RBBK01] Ph. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001. 12, 32, 46, 80, 83

[Ree54] I. Reed. A Class of Multiple-Error-Correcting Codes and the Decoding Scheme. *IRE Transactions on Information Theory*, (4):38–49, September 1954. 124

[Reg06] O. Regev. Lattice-Based Cryptography. In *Advances in Cryptology - CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 131–141. Springer Berlin Heidelberg, 2006. 30

[Reg10] O. Regev. The Learning with Errors Problem (Invited Survey). In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity - CCC'10*, pages 191–204, 2010. 101

[rfc06] The Transport Layer Security (TLS) Protocol. IETF RFC 4253 (Informational), January 2006. 42

[rfc08] The Secure Shell (SSH) Transport Layer Protocol. IETF RFC 5246 (Informational), August 2008. 42

[Riv] R. L. Rivest. The MD5 Message-Digest Algorithm (RFC 1321). `www.ietf.org/rfc/rfc1321.txt?number=1321`. 39

[Rog02] Ph. Rogaway. Authenticated-Encryption with Associated-Data. In *ACM Conference on Computer and Communications Security*, pages 98–107, 2002. 45, 46, 80, 83

[Rog04a] Ph. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT'04*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004. 11, 20, 45, 46, 80, 84, 93

[Rog04b] Ph. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT'04*, pages 16–31, 2004. 83

[Ros38] J. B. Rosser. The n-th Prime is Greater than $n \ln n$. In *Proceedings of the London Mathematical Society*, volume 45, pages 21–44, 1938. 130

[RS60] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. 124

[RS06] Ph. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In *Advances in Cryptology - EUROCRYPT'06*, volume 4341 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006. 44, 46, 80

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. 11, 12, 22, 50, 106

[RST01] R. L. Rivest, A. Shamir, and Y. Tauman. How to Leak a Secret. In *Advances in Cryptology - ASIACRYPT'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001. 50, 59

[RVV15] R. Reyhanitabar, S. Vaudenay, and D. Vizár. Boosting OMD for Almost Free Authentication of Associated Data. *IACR Cryptology ePrint Archive*, 2015:302, 2015. 95

[SA03] S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'02*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2003. 146

[saf] SAFEcrypto. www.safecrypto.eu. 30

[Sch90] C.-P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology - CRYPTO'89*, volume 434 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1990. 51, 60

[SE12] M.-J. O. Saarinen and D. W. Engels. A Do-It-All-Cipher for RFID: Design Requirements (Extended Abstract). *IACR Cryptology ePrint Archive*, 2012(317), 2012. 99

[Sec91] RSA Data Security. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications, 1991. 50

[Seu12] Y. Seurin. On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model. In *Advances in Cryptology - EUROCRYPT'02*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2012. 61

[Sha48] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, 1948. 23, 124

[Sha49] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, 1949. 11, 20, 23

[Sha84] A. Shamir. A Polynomial-Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. *IEEE Transactions on Information Theory*, 30(5):699–704, 1984. 11, 22

[Sho97] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997. 30

[Shp06] I. E. Shparlinski. On RSA Moduli with Prescribed Bit Patterns. *Designs, Codes and Cryptography*, 39(1):113–122, 2006. 106, 113

[SIH+11] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *Proceedings of the 13th International Workshop on Cryptographic Hardware and Embedded Systems – CHES'11*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, 2011. 100

[Sim97] D. R. Simon. On the Power of Quantum Computation. *SIAM Journal on Computing*, 26:1474–1483, 1997. 30

[Sin99] S. Singh. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography*. Doubleday, 1st edition, 1999. 12, 22

[SKD+15] T. Sato, D. M. Kammen, B. Duan, M. Macuha, Z. Zhou, J. Wu, M. Tariq, and S. A. Asfaw. *Smart Grid Standards: Specifications, Requirements, and Technologies*, chapter Future of the Smart Grid, pages 379–393. John Wiley & Sons Singapore Pte. Ltd, 2015. 42

[SL07] M. Singh and L. C. Lau. Approximating Minimum Bounded Degree Spanning Trees to within One of Optimal. In *Proceedings of the 39th annual ACM symposium on Theory of computing*, pages 661–670. ACM, 2007. 73

[Sma01] N. P. Smart. An Identity Based Authenticated Key Agreement Protocol Based on the Weil Pairing. *Electronics Letters*, 38:630–632, 2001. 56

[SOK00] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems Based on Pairings. In *Symposium on Cryptography and Information Security - SCIS'00-c20*, Lecture Notes in Computer Science, 2000. 56

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7(3-4):281–292, 1971. 132

[Ste87] J. Stern. Secret Linear Congruential Generators Are Not Cryptographically Secure. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science - FOCS'87*, pages 421–426. IEEE Computer Society, 1987. 10, 20

[Ste12] W. A. Stein. Sage Mathematics Software (Version 5.0), 2012. 151

[TAL93] R. Tolimieri, M. An, and C. Lu. *Mathematics of Multidimensional Fourier Transform Algorithms*. Springer, 1993. 116

[Tat58] J. Tate. WC-Groups Over p-Adic Fields. *Séminaire Bourbaki*, 4:265–277, 1956-1958. 56

[Tat63] J. Tate. Duality Theorems in Galois Cohomology Over Number Fields. In *Proceedings of the International Congress of Mathematicians - Stockholm'62*, Lecture Notes in Computer Science, page 288–295, 1963. 56

[Too63] A. L. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. *Soviet Mathematics Doklady*, 3:714–716, 1963. 132

[Tur36] A. Turing. On Computable Numbers with an Application to the " Entscheidungsproblem". *Proceeding of the London Mathematical Society*, pages 230–265, 1936. 26

[UMS11] S. K. Udgata, A. Mubeen, and S. L. Sabat. Wireless Sensor Network Security Model Using Zero Knowledge Protocol. In *ICC*, pages 1–5. IEEE, 2011. 72

[Val91] B. Vallée. Gauss' Algorithm Revisited. *Journal of Algorithms*, 12(4):556–572, 1991. 125, 130, 152

[Van92] S. A. Vanstone. Responses to NIST's Proposal. *Communications of the ACM*, 35:50–52, 1992. 152

[Vau02] S. Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In *Advances in Cryptology - EUROCRYPT'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, 2002. 42, 45, 80

[Ver26] G. S. Vernam. Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications. *Transactions of the American Institute of Electrical Engineers*, XLV, 1926. 20

[vM39] R. von Mises. Über Aufteilungs-und Besetzungs-Wahrscheinlichkeiten. *Revue de la Faculté des Sciences de l'Université d'Istanbul*, (4):145–163, 1939. 39

[vN51] J. von Neumann. Various Techniques used in Connection with Random Digits. *National Bureau of Standards Applied Math Series*, 12:36–38, 1951. 138

[VOW96a] P. C. Van Oorschot and M. J. Wiener. Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude. In *Advances in Cryptology - CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 1996. 20

[vOW96b] P. C. van Oorschot and M. J. Wiener. On Diffie-Hellman Key Agreement with Short Exponents. In *Advances in Cryptology — EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 1996. 52

[vOW99] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. volume 12, pages 1–28. Springer, 1999. 39

[VPH+11] K. R. Venugopal, L. M. Patnaik, M. Hashim, Santhosh Kumar G., and A. Sreekumar. Authentication in Wireless Sensor Networks Using Zero Knowledge Protocol. In *Computer Networks and Intelligent Computing*, volume 157, pages 416–421. Springer Berlin Heidelberg, 2011. 72

[vT91] J. van Tilbourg. Method for the Modular Reduction of numbers, 1991. filed October 2, 1991. 104

[VZ95] S. A. Vanstone and R. J. Zuccherato. Short RSA Keys and Their Generation. *Journal of Cryptology*, 8(2):101–114, 1995. 104, 106

[WBYD00] H. Wu, F. Bao, D. Ye, and R. H. Deng. Cryptanalysis of the m-Permutation Protection Schemes. In *Australasian Conference on Information Security and Privacy - ACISP'00*, volume 1841, pages 97–111. Springer, 2000. 147

[Wei40] A. Weil. Sur les fonctions algébriques à corps de constantes fini. *Comptes rendus de l'Académie des Sciences*, 210:592–594, 1940. 56

[WH99] H. Wu and M. A. Hasan. Closed-Form Expression for the Average Weight of Signed-Digit Representations. *IEEE Transactions on Computers*, 48(8):848–851, 1999. 132

[WHF03] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). IETF RFC 3610 (Informational), September 2003. 43

[WP03] X. Wang and V. Y. Pan. Acceleration of Euclidean Algorithm and Rational Number Reconstruction. *SIAM Journal on Computing*, 32(2):548–556, 2003. 152

[Wri87] P. Wright. *Spycatcher: The Candid Autobiography of a Senior Intelligence Officer*. Mandarin, 1987. 45

[Wu08] H. Wu. The Stream Cipher HC-128. In *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 39–47. Springer-Verlag, 2008. 10, 20

[Yao86] A. C.-C. Yao. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, October 1986. 59

[YK05] M. Yung and J. Katz. *Digital Signatures (Advances in Information Security)*. Springer-Verlag, 2005. 54

[Yuv79] G. Yuval. How to Swindle Rabin. *Cryptologia*, 3:187–189, 1979. 39

[ZPS92] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL - A One-Way Hashing Algorithm with Variable Length of Output. In *AUSCRYPT*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104. Springer, 1992. 39

[ZQWZ10] L. Zhang, B. Qin, Q. Wu, and F. Zhang. Efficient Many-to-One Authentication with Certificateless Aggregate Signatures. *Computer Networks*, 54(14):2482–2491, 2010. 72

[ZW09] H. Zhang and X. Wang. Cryptanalysis of Stream Cipher Grain Family. *IACR Cryptology ePrint Archive*, 2009:109, 2009. 20

CHAPTER 7

LIST OF MAIN ABBREVIATIONS

AE	Authenticated Encryption
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-GCM	Advanced Encryption Standard in Galois Counter Mode
AES-NI	Advanced Encryption Standard Instruction Set
AC-Message	Authentication Challenge Message
A-Message	Authentication Message
ANSI	American National Standards Institute
AR-Message	Authentication Request Message
AVR	Alf and Vegard RISC processor
BCH	Bose-Chaudhuri-Hocquenghem
BDDHP	Bilinear Decisional Diffie-Hellman Problem
BDHP	Bilinear Diffie-Hellman Problem
BEAST	Browser Exploit Against SSL/TLS
BPP	Binary Packet Protocol
CA	Certificate Authority
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CDHP	Computational Diffie-Hellman Problem
CFB	Cipher Feedback
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTR	Counter
DDHP	Decisional Diffie-Hellman Problem
DES	Data Encryption Standard
DESL	Data Encryption Standard Lighweight
DESXL	Data Encryption Standard Extra-Lighweight
Dlog	Discrete logarithm
DLP	Discrete Logarithm Problem
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECADD	Elliptic Curve (Point) Addition

ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography
ECC	Error Correcting Code
ECDBL	Elliptic Curve (Point) Doubling
ECDDH	Elliptic Curve Decisional Diffie-Hellman
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECMQV	Elliptic Curve Menezes-Qu-Vanstone
ECSM	Elliptic Curve Scalar Multiplication
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
ERP	The e-th Root Problem
FACT	Factoring
FIB	Focused Ion Beam
FTG	Find then Guess
GCQH	The Government Communications Headquarters
GC&QS	The Government Code and Cypher School
GCM	Galois Counter Mode
GE	Gate Equivalent
GPS	Girault-Poupard-Stern
HDL	Hardware Description Language
HMAC	Keyed-Hash Message Authentication Code
HRP	The Higher Residuosity Problem
IND	Indistinguishability
IND-CCA1	Indistinguishability under Chosen Ciphertext Attack
IND-CCA2	Indistinguishability under Adaptive Ciphertext Attack
IND-CPA	Indistinguishability under Chosen Plaintext Attack
INT-CTXT	Integrity of Ciphertext
IoT	Internet of Things
IQR	Interquartile Range
IV	Initialization Vector
LFSR	Linear Feedback Shift Register
LLL	Lenstra–Lenstra–Lovász
LOR	Left or Right Indistinguishability
LOR-CCA	Left or Right Indistinguishability under Chosen Ciphertext Attack
LOR-CPA	Left or Right Indistinguishability under Chosen Plaintext Attack
LPN	Learning Parity with Noise
LSB	Least Significant Bit
LWE	Learning With Errors
MAC	Message Authentication Code
MAC-FORGE	Message Authentication Code - Forge
MD	Message Digest
MSB	Most Significant Bit
NAND	Negative-AND
NESSIE	New European Schemes for Signatures, Integrity and Encryption
NIST	National Institute of Standards and Technology
NLFSR	Non-Linear Feedback Shift Register

NM	Non-Maleability
NM-CCA	Non-Maleability under Chosen Ciphertext Attack
NTRU	n-th degree TRUncated polynomial ring
OCB	Offset Codebook Mode
OFB	Output Feedback
OMD	Offset Merkle-Damgård
PKI	Public Key Infrastructure
p-OMD	Pure Offset Merkle-Damgård
PPT	Probabilistic Polynomial Time
PRF	Pseudorandom Function
PRP	Pseudorandom Permutation
QRP	Quadratic Residuosity Problem
RFC	Request for Comments
RFID	Radio Frequency Identification
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
R-M	Reed-Muller
RO	Random Oracle
ROM	Random Oracle Model
ROR	Real or Random Indistinguishability
ROR-CCA	Real or Random Indistinguishability under Chosen Ciphertext Attack
ROR-CPA	Real or Random Indistinguishability under Chosen Plaintext Attack
ROTR	Rotate Right
RSA	Rivest-Shamir-Adleman
RUP	Releasing Unverified Plaintext
SCA	Side Channel Attacks
SCM	Single Constant Multiplication
SEM	Semantic Security
SHA	Secure Hash Algorithm
SHR	Right Shift
SIV	Synthetic Initialization Vector
SPA	Simple Power Analysis
SPN	Substitution-Permutation Network
SSH	Secure Shell
SSL/TLS	Secure Sockets Layer/Transport Layer Security
SU-KMA	Strong Unforgeability Under Known-Message Attack
SU-RMA	Strong Unforgeability Under Random-Message Attack
TTP	Trusted Third Party
UV	Ultraviolet
VM	Virtual Machine
XEX	Xor-Encrypt-Xor
XMACC	Counter-Based XOR MAC
XOR	Exclusive OR
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing
ZADDC	Conjugate Addition in co-Z
ZADDU	Addition and Update in co-Z
ZK	Zero-Knowledge
ZKP	Zero-Knowledge Proof/Protocol

APPENDIX A

COMPRESSION FUNCTIONS

A.1 Compression Functions of SHA-256 and SHA-512

We recall the compression functions of the standard SHA-256 and SHA-512 hash functions following NIST FIPS PUB 180-4 [FIP12]. We denote the underlying compression functions of these standard hash functions by sha-256 and sha-512, respectively.

Note. In the following, by *word* we mean a group of either 32 bits (4 bytes) or 64 bits (8 bytes), depending on the compression function algorithm. Namely, in sha-256 each word is a 32-bit string and in sha-512 each word is a 64-bit string.

$\mathbf{ROTR^n(x)}$: The *rotate right* (circular right shift) operation, where x is a w-bit word and n an integer with $0 \leq n < w$, is defined by $\text{ROTR}^n(x) = (x \gg n) \vee (x \ll w - n)$

$\mathbf{SHR^n(x)}$: The *right shift* operation, where x is a w-bit word and n an integer with $0 \leq n < w$, is defined by $\text{SHR}^n(x) = (x \gg n)$.

Choice Function. Let m be 32 in the case of sha-256 and 64 in the case of sha-512. The *choice function* takes as input two m-bit words y and z, and one m-bit word x selector input, and returns an m-bit word. Every value of a single bit of x is used to select one of the bit of the m pairs (from (y_0, y_1) to (z_{m-2}, z_{m-1})). It is similar to an m-to-$m/2$ multiplexer (*i.e.* m 2-to-1 multiplexers). One can use indifferently whether inclusive OR ($\vee$) or exclusive OR ($\oplus$). The function is defined as follows:

$$\text{Ch} : \left| \begin{array}{rcl} \{0,1\}^m \times \{0,1\}^m \times \{0,1\}^m & \longrightarrow & \{0,1\}^m \\ x, y, z & \longmapsto & (x \wedge y) \oplus (\neg x \wedge z) \end{array} \right.$$

Majority Function. Let m be 32 in the case of sha-256 and 64 in the case of sha-512. In Boolean logic, the *majority function* (also called the median operator) is a function from n inputs to one output. The value of the operation is false when $n/2$ or more arguments are false, true otherwise. In sha-256/sha-512 design, *majority function* takes as input three m-bit words x, y, z, and returns an m-bit word. Such as *choice function*, one can use indifferently whether inclusive OR ($\vee$) or exclusive OR ($\oplus$). The function is defined as follows:

$$\text{Maj} : \left| \begin{array}{rcl} \{0,1\}^m \times \{0,1\}^m \times \{0,1\}^m & \longrightarrow & \{0,1\}^m \\ x, y, z & \longmapsto & (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \end{array} \right.$$

A.1.1 The Compression Function of SHA-256

Sigma Functions. The $\Sigma_0^{\{256\}}$ and $\Sigma_1^{\{256\}}$ functions are respectively represented by a multiplication by the $X^2+X^{13}+X^{22}$ and $X^6+X^{11}+X^{25}$ polynomials of $\mathbb{F}_2[X]/X^{32}+1$, if one represents any 32-bit word $W=(W[0]W[1]...W[31])$ as a $\mathbb{F}_2[X]/X^{32}+1$ polynomial $W[0]+W[1].X+W[2].X^2+...+W[31].X^{31}$. The functions as are follows:

$$\Sigma_0^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) \end{array} \right.$$

$$\Sigma_1^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) \end{array} \right.$$

The $\sigma_0^{\{256\}}$ and $\sigma_1^{\{256\}}$ functions are respectively represented by a multiplication by the X^7+X^{18} and $X^{17}+X^{19}$ polynomials of $\mathbb{F}_2[X]/X^{32}+1$, if one represents any 32-bit word $W=(W[0]W[1]...W[31])$ as a $\mathbb{F}_2[X]/X^{32}+1$ polynomial $W[0]+W[1].X+W[2].X^2+...+W[31].X^{31}$. These multiplications are applied on the quotient of the Euclidean division of the polynomial representation of a 32-bit word W, and the polynomial $P=X$. The functions as are follows:

$$\sigma_0^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) \end{array} \right.$$

$$\sigma_0^{\{256\}} \left| \begin{array}{rcl} \{0,1\}^{32} & \longrightarrow & \{0,1\}^{32} \\ x & \longmapsto & \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x) \end{array} \right.$$

The Process. The sha-256 function process is defined as below:

$$\mathrm{sha}-256 \left| \begin{array}{rcl} \{0,1\}^{256} \times \{0,1\}^{512} & \longrightarrow & \{0,1\}^{256} \\ H, M & \longmapsto & C \end{array} \right.$$

Let H be the 256-bit *hash input* (chaining input) and M be the 512-bit *message input*. These two inputs are represented respectively by an array of eight 32-bit words $H_0, \cdots, H_7$ and an array of sixteen 32-bit words $M_0, \cdots, M_{15}$. The 256-bit output value C is also represented as an array of eight 32-bit words $C_0, \cdots, C_7$.

During the process of compression, a sequence of 64 constant 32-bit words, $K_0^{\{256\}}, ..., K_{63}^{\{256\}}$ are used. These 32-bit words represent the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers. We refer the reader to [FIP12] for a table containing these constants.

Furthermore, addition ($+$) is performed modulo 2^{32}.

The compression function processes is detailed in Algorithm 31.

Algorithm 31: Compression function of SHA-256

for $t \leftarrow 0$ **to** 15 **do**
  $W_t = M_t$
end for
for $t \leftarrow 16$ **to** 63 **do**
  $\sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}$
end for
$(a, b, c, d, e, f, g, h) \leftarrow (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$
for $t \leftarrow 0$ **to** 63 **do**
  $T_1 \leftarrow h + \Sigma_1^{\{256\}}(e) + \mathrm{Ch}(e, f, g) + K_t^{\{256\}} + W_t$
  $T_2 \leftarrow \Sigma_0^{\{256\}}(a) + \mathrm{Maj}(a, b, c)$
  $(h, g, f, e, d, c, b, a) \leftarrow (g, f, e, d + T_1, c, b, a, T_1 + T_2)$
end for
$(C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7) \leftarrow (a + H_0, b + H_1, c + H_2, d + H_3, e + H_4, f + H_5, g + H_6, h + H_7)$

A.1.2 The Compression Function of SHA-512

Sigma Functions. The $\Sigma_0^{\{512\}}$ and $\Sigma_1^{\{512\}}$ functions are respectively represented by a multiplication by the $X^{28}+X^{34}+X^{39}$ and $X^{14}+X^{18}+X^{41}$ polynomials of $\mathbb{F}_2[X]/X^{64}+1$, if one represents any 64-bit word $W = (W[0]W[1]...W[63])$ as a $\mathbb{F}_2[X]/X^{64}+1$ polynomial $W[0] + W[1] \cdot X + W[2] \cdot X^2 + ... + W[63] \cdot X^{63}$. The functions are as follows:

$$\Sigma_0^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{ROTR}^{28}(x) \oplus \mathrm{ROTR}^{34}(x) \oplus \mathrm{ROTR}^{39}(x) \end{array} \right.$$

$$\Sigma_1^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{ROTR}^{14}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{ROTR}^{41}(x) \end{array} \right.$$

The $\sigma_0^{\{512\}}$ and $\sigma_1^{\{512\}}$ functions are respectively represented by a multiplication by the $X^1 + X^8$ and $X^{19} + X^{61}$ polynomials of $\mathbb{F}_2[X]/X^{64} + 1$, if one represents any 64-bit word $W = (W[0]W[1]...W[63])$ as a $\mathbb{F}_2[X]/X^{64} + 1$ polynomial $W[0] + W[1] \cdot X + W[2] \cdot X^2 + ... + W[63] \cdot X^{63}$. These multiplications are applied on the quotient of the Euclidean division of the polynomial representation of a 64-bit word W, and the polynomial $P = X$. The functions as are follows:

$$\sigma_0^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{ROTR}^{1}(x) \oplus \mathrm{ROTR}^{8}(x) \oplus \mathrm{SHR}^{7}(x) \end{array} \right.$$

$$\sigma_0^{\{512\}} \left| \begin{array}{rcl} \{0,1\}^{64} & \longrightarrow & \{0,1\}^{64} \\ x & \longmapsto & \mathrm{SHR}^{19}(x) \oplus \mathrm{SHR}^{61}(x) \oplus \mathrm{SHR}^{6}(x) \end{array} \right.$$

The Process. The sha-512 compression function is defined as below:

$$\mathrm{sha}-512 \left| \begin{array}{rcl} \{0,1\}^{512} \times \{0,1\}^{1024} & \longrightarrow & \{0,1\}^{512} \\ H, M & \longmapsto & C \end{array} \right.$$

Let M be the 1024-bit *message input* and H the 512-bit *hash input* (chaining input). These two inputs are represented respectively by an array of sixteen 64-bit words $M_0, \cdots, M_{15}$, and an array of eight 64-bit words $H_0, \cdots, H_7$. The 512-bit output value C is also represented as an array of eight 64-bit words $C_0, \cdots, C_7$.

Let H be the 512-bit *hash input* (chaining input) and M be the 1024-bit *message input*. These two inputs are represented respectively by an array of 8 64-bit words $H_0, \cdots, H_7$ and an array of 16 64-bit words $M_0, \cdots, M_{15}$. The 512-bit output value C is also represented as an array of 8 64-bit words $C_0, \cdots, C_7$.

During the process of compression, a sequence of 80 constant 64-bit words $K_0^{\{512\}}, ..., K_{79}^{\{512\}}$ is used. These 64-bit words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. We refer the reader to [FIP12] for a table containing these constants.

Addition ($+$) is performed modulo 2^{64}.

The compression function is described in Algorithm 32.

Algorithm 32: Compression function of SHA-512

for $t \leftarrow 0$ **to** 15 **do**
  $W_t = M_t$
end for
for $t \leftarrow 16$ **to** 79 **do**
  $W_t = \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16}$
end for
$(a, b, c, d, e, f, g, h) \leftarrow (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$
$(C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7) \leftarrow (a + H_0, b + H_1, c + H_2, d + H_3, e + H_4, f + H_5, g + H_6, h + H_7)$

APPENDIX B

FAULT ATTACKS ON ECC: CO-Z FORMULÆ

Let $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$ be two points of $E^{\mathcal{J}}(\mathbb{F}_p)$ with $P \neq \pm Q$. Addition and update in co-Z (ZADDU) is the procedure to compute $P + Q$ and update the point P to feature the same Z-coordinate (see [Mel07]). Conjugate addition in co-Z (ZADDC) is the procedure to compute $P + Q$ and $P - Q$ (see [GJM10]).

$$\text{Algorithm ZADDU} = \begin{cases} C & \leftarrow & (X_1 - X_2)^2 \\ W_1 & \leftarrow & X_1 C \\ W_2 & \leftarrow & X_2 C \\ Z_3 & \leftarrow & Z(X_1 - X_2) \\ D & \leftarrow & (Y_1 - Y_2)^2 \\ A_1 & \leftarrow & Y_1(W_1 - W_2) \\ X_3 & \leftarrow & D - W_1 - W_2 \\ Y_3 & \leftarrow & (Y_1 - Y_2)(W_1 - X_3) - A_1 \\ R & := & (X_3, Y_3, Z_3) \\ S & := & (W_1, A_1, Z_3) \qquad \text{return}(R = P + Q, S) \end{cases}$$

$$\text{Algorithm ZADDC} = \begin{cases} C & \leftarrow & (X_1 - X_2)^2 \\ W_1 & \leftarrow & X_1 C \\ W_2 & \leftarrow & X_2 C \\ Z_3 & \leftarrow & Z(X_1 - X_2) \\ D_1 & \leftarrow & (Y_1 - Y_2)^2 \\ A_1 & \leftarrow & Y_1(W_1 - W_2) \\ X_3 & \leftarrow & D_1 - W_1 - W_2 \\ Y_3 & \leftarrow & (Y_1 - Y_2)(W_1 - X_3) - A_1 \\ D_2 & \leftarrow & (Y_1 + Y_2)^2 \\ X_4 & \leftarrow & D_2 - W_1 - W_2 \\ Y_4 & \leftarrow & (Y_1 + Y_2)(W_1 - X_4) - A_1 \\ R & := & (X_3, Y_3, Z_3) \\ S & := & (X_4, Y_4, Z_3) \qquad \text{return}(R = P + Q, S = P - Q) \end{cases}$$

Algorithm 33: Add only Montgomery ladder using co-Z formulæ [GJM10]

Input: a point P and an integer $k = (1, k_{N-2}, k_{N-3,} \ldots, k_0)_2$

Output: $[k]P$

1 $R_0 \leftarrow P$

2 $R_1 \leftarrow 2P$

3 **for** $i = N - 2$ downto 0 **do**

4 $\quad (R_{1-k_i}, R_{k_i}) \leftarrow \text{ZADDC}(R_{k_i}, R_{1-k_i})$

5 $\quad (R_{k_i}, R_{1-k_i}) \leftarrow \text{ZADDU}(R_{1-k_i}, R_{k_i})$

6 **end for**

7 **return** R_0

APPENDIX C

DETERMINISTIC SIGNATURE SCHEME

This section recalls the provably-secure deterministic signature scheme of [MNPV98]. The scheme initially uses a subgroup of order q of the multiplicative group $\mathbb{F}_p^*$. We adapted it to ECCs.

The scheme uses the following curve parameters:
– $E\colon y^2 = x^3 + ax + b$, an elliptic curve over some prime base field $\mathbb{F}_p$ with parameters a, b
– $G = (x_G, y_G)$, a generator of a subgroup of E of order n
The private key is an integer $d \in_R [1, n-1]$. The corresponding public key is $P = (x_P, y_P) = [d]G$.

Algorithm 34: Sign

Input: Private key d, message m, hash function h

Output: Signature (e, s)

$u \leftarrow h(d, m, p, a, b, n, G, P)$
$Q \leftarrow [u]G$
$r \leftarrow x_Q \bmod n$
$e \leftarrow h(m, r) \bmod n$
$s \leftarrow u - de \bmod n$
return (e, s)

Algorithm 35: Verify

Input: Public key P, message m, signature (e, s), hash function h

Output: `True` or `False`

$Q \leftarrow [s]G + [e]P$
$r \leftarrow x_Q \bmod n$
if $e = h(m, r) \bmod n$ **then**
    return `True`
end if
else
    return `False`
end if

Appendix D

Code: Barrett's Algorithm for Polynomials

$p_1(x) = \sum_{i=0}^{7}(10+i)x^i$ and $p_2(x) = x^3 + x^2 + 110$

```
(define p1 '((7 17) (6 16) (5 15) (4 14) (3 13) (2 12) (1 11) (0 10)))
(define p2 '((3 1) (2 1) (0 110)))
```

;shifting a polynomial to the right

```
(define shift (lambda (l q)
(if (or (null? l) (< (caar l) q)) '() (cons (cons (- (caar l) q) (cdar l))
(shift (cdr l) q)))))
```

;adding polynomials

```
(define add (lambda (p q)
(degre (if (>= (caar p) (caar q)) (cons p (list q)) (add q p)))))
```

;multiplying a term by a polynomial, without monomials $\prec x^{lim}$

```
(define txp (lambda (terme p lim)
(if (or (null? p) (> lim (+ (car terme) (caar p)))) '() (cons (cons (+ (car terme)
(caar p)) (list (* (cadr terme) (cadar p)))) (txp terme (cdr p) lim)))))
```

;multiplying a polynomial by a polynomial, without monomials $\prec x^{lim}$

```
(define mul (lambda (p1 p2 lim)
(if p1 (cons (txp (car p1) p2 lim) (mul (cdr p1) p2 lim)) '())))
```

;management of the exponents

```
(define sort (lambda (p n)
(if p (+ ((lambda(x) (if x (cadr x) 0)) (assoc n (car p))) (sort (cdr p) n)) 0)))
(define order (lambda (p n)
(if(> 0 n) '() (let ((factor (sort p n))) (if (not (zero? factor))
(cons (cons n (list factor)) (order p (-n 1))) (order p (-n 1)))))))
(define degre (lambda(p) (order p ((lambda(x)(if x x -1)) (caaar p)))))
```

;Euclidean division

```
(define divide (lambda (q p r)
(if (and p (<= (caar p) (caar q))) (let ((tampon (cons (- (caar q)(caar p))
(list (/ (cadar q) (cadar p)))))) (divide (add (map (lambda(x) (cons (car x)
(list (-cadr x))))(txp tampon p -1)) q) p (cons tampon r))) (reverse r)))
(define division (lambda (q p) (divide q p '())))
```

;Barrett(k, L, last_P and Y representing K, L, P and $\mathbf{y}$*)*

```
(define k)
(define y)
(define L 8)
```

```
(define last_P)

(define barrett (lambda (q p)

(if (eq ? last_P p) (letrec ((g (caar q)) (h (- (+ g 1) y))) (shift (degre (mul (shift k (-L g 1)) (shift q
y) h)) h)) (begin (set!  k (division (list (cons L '(1) )) p)) (set!  y (caar (set!  last_P p))) (barrett q
p))))
```

Appendix E

Code: Backtracking Assisted Multiplication

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int False = 0;
int True = 255;

#define step(u,v) path[dep+1][0]=u; path[dep+1][1]=v; backtracking(dep+1);
#define steps(x,n) c=x; if(c<256) {if (visited[c]==False) {visited[c]=True;\
        path[dep][2]=c;path[dep][3]=n;step(a,b);step(a,c);step(b,c);visited[c]=False;}}

int visited[2 * 256];
int maxdep,path[256][4],maxpath[256][4];
int size;

void backtracking(int dep){

    if (dep > maxdep){
        maxdep = dep;
        memcpy(maxpath, path, sizeof(path));
    }

    int a = path[dep][0];
    int b = path[dep][1];
    int c;

    steps(a+b,         1);
    steps(a<<1,        2);
    steps(b<<1,        3);
    steps((a<<1)%256, 4);
    steps((b<<1)%256, 5);
    steps((a+b)%256,  6);
    steps(abs(a-b),    7);
}

int main() {
    int i,j;
    FILE *Fin, *Fout;
    int *A;

    Fin = fopen("input.txt", "r");
    fscanf(Fin, "%d", &size);
    A = (int*)malloc(sizeof(int)*size);
    for (i=0; i <size; ++i) {
        fscanf(Fin, "%d", &A[i]);
    }

    for (i=0; i<256; ++i)  visited[i] = True;
    for (i=0; i<size; ++i) visited[A[i]] = False;

    free(A);

    maxdep = -1;
    for (i=0; i<256; ++i) {
        for (j=i+1; j<256; ++j) {
            if (visited[i]==True || visited[j]==True) continue;
            path[0][0] = i;
            path[0][1] = j;
            visited[i] = visited[j] = True;
            backtracking(0);
            visited[i] = visited[j] = False;
        }
    }

    Fout = fopen("output.txt","w");
    for (i=0; i < maxdep; ++i)
        fprintf(Fout,"%3d %3d %3d %3d\n",
                maxpath[i][0],  // a_i
                maxpath[i][1],  // a_j
                maxpath[i][2],  // a_p
                maxpath[i][3]); // op

    return EXIT_SUCCESS;
}
```

Appendix F

Code: Regulating the Pace of von Neumann Correctors

```
import random
import numpy
import math

# Available memory
m = 1000

# Distributional regulator
mu_D = lambda x:icdf(1 - x/m)

def unif_icdf(x):
    """
    Inverse cumulative distribution function for the uniform distribution
    U(a, b)
    """
    a = 200
    b = 600
    return a + x * (b-a)

def generator(icdf):
    """
    Generates a random number distributed according to the provided
    inverse cumulative distribution function
    """
    return icdf(random.random())

def simulate(input_events, mu):
    """
    Simulation
        input_events: relative time between input events
        mu: regulator
    """

    # Stack population
    X = 0

    # Current input
    k = 0

    # Lookahead
    j = 0

    # Absolute time for output events
    M = []

    # Compute absolute time for input events
    T = [0] * len(input_events)
    for k in range(1, len(input_events)):
        T[k] = T[k-1] + input_events[k]

    # Push the first input
    X += 1

    while k+j+1 < len(input_events) - 1:
        j = 0
        # Push all early inputs on stack
        while T[k+j+1] < M[-1]:
            X+=1
            j+=1

        # Memory overflow or underflow
        if X < 0 or X >= m:
            print("Error! Memory under- or overflow: X = %s"%X)
            return []

        # Pop and emit an object
        M.append(M[-1] + mu(X))
        X -= 1
        k += j

    return M

def save_data(ret, filename):
    """
    Saves data ret to the file 'filename'
    """
    f = open(filename,'w')
    f.write("%s\n"%("mu"))
    for u in ret:
        a = u
        f.write("%s\n"%(a))

    f.close()
```

```
def generate_events(N,icdf):
    """
    Generates N events distributed according to the provided
    inverse cumulative distribution function
    """
    return [generator(icdf) for i in range(N)]


events = generate_events(100000,unif_icdf)
ret = simulate(events, mu_D(unif_icdf))
save_data(ret, 'output.txt')
```

Résumé

Cette thèse aborde différents aspects de la cryptologie, subsumant des champs aussi variés que la conception de protocoles, l'amélioration d'outils algorithmiques et les attaques. Les deux principales focales de cette étude sont: un protocole de co-signature prouvé irréfragable et un système de chiffrement authentifié à sécurité prouvée. Notre protocole de co-signature permet l'équité légale. L'équité légale est une nouvelle variante de la notion d'équité, ne reposant pas sur des tiers. Notre instanciation d'équité légale est construite à l'aide des signatures de Schnorr. Nous présenterons également un protocole d'authentification distribué de type Fiat-Shamir. La deuxième partie de cette thèse est consacrée aux améliorations algorithmiques. Nous introduisons une méthode permettant de doubler la vitesse de l'algorithme de Barrett en utilisant des modules composites spécifiques et un nouvel algorithme de multiplication à retour sur trace, particulièrement adapté aux microprocesseurs bon marché. Nous nous intéresserons ensuite à la sécurité des composants en étudiant la régulation du débit des correcteurs de von Neumann et les attaques en fautes sur des implémentations de cryptographie à courbes elliptiques. Enfin, un des actes novatoires incidents notre travail sera d'adapter aux codes correcteurs d'erreurs deux techniques empruntées à la cryptographie: un premier résultat améliore l'efficacité calculatoire des codes BCH grâce à une version de l'algorithme de Barrett étendue aux polynômes. Le second est un nouveau code correcteur d'erreurs basé sur la théorie des nombres.

Mots-clés: cryptographie, authentification, chiffrement, équité, arithmetique, attaques, codes correcteurs d'erreurs.

Abstract

This thesis addresses various topics in cryptology, namely protocol design, algorithmic improvements and attacks. In addition, we venture out of cryptography and propose two new applications of cryptographic techniques to error correcting codes.
Our main results comprise a provably secure co-signature protocol and a provably secure authenticated encryption scheme. Our co-signature protocol achieves *legal fairness*, a novel fairness variant that does not rely on third parties. Legal fairness is implemented using Schnorr signatures. We also present a distributed Fiat-Shamir authentication protocol.
The second part of the thesis is devoted to computational improvements, we discuss a method for doubling the speed of Barrett's algorithm by using specific composite moduli, devise new BCH speed-up strategies using polynomial extensions of Barrett's algorithm, describe a new backtracking-based multiplication algorithm suited for lightweight microprocessors and present a new number theoretic error-correcting code.
Fault injection attacks are further overviewed and a new fault attack on ECC implementations is proposed.

Keywords: cryptography, authentication, encryption, fairness, arithmetic, attacks, error correcting codes.