A Novel Asymmetric Scheme with Stream Cipher Construction

Jean-Philippe Aumasson

Master's thesis

September 2006

Responsible:
Prof. Serge Vaudenay
LASEC, EPFL

LASEC

"Tout ce qui a pu se dire contre la science ne saurait faire oublier que la recherche scientifique reste, dans la dégradation de tant d'ordres humains, l'un des rares domaines ou l'homme se contrôle, s'incline devant le raisonnable, est non bavard, non violent et pur. Moments de la recherche certes constamment interrompus par les banalités du quotidien mais qui se renouent en durée propre. Le lieu de la morale et de l'élévation ne se trouve-t-il pas désormais au laboratoire ?"

Emmanuel Levinas (Le Monde, 19/20 mars 1978)

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Serge Vaudenay, his outstanding care and concern for students is exemplary. I am also grateful to Matthieu Finiasz for his assistance and many LaTeX tips.

I address many thanks to Willi Meier for his relevant remarks and kindness, and to his student Simon Kűnzli for fruitful discussions about TCHO.

My officemates Salvatore Bocchetti, Jean Monnerat, Florin Oswald, and Raphael Phan shall also be mentionned here, for their sporadic help and constant sympathy.

On the personal side, I am deeply indebted to my parents for making this expatriation possible, and finally my most tender thanks go to Paula, for her support.

Abstract

This work is based on the public-key stream cipher TCHO designed by Finiasz and Vaudenay, which relies on the hardness of finding a low-weight multiple of a given high-degree polynomial over the field $\mathbb{F}_2$ of arbitrary weight, and on the noisy decoding of the linear code spanned by a linear feedback shift register (LFSR). The encryption procedure is non-deterministic: it involves two LFSR's, and a source of random bits of a given bias, whereas decryption consists in an exhaustive search algorithm and simple linear algebra operations.

Until now, stream ciphers were only symmetric, and asymmetric schemes were somewhat difficult to employ in constrained environments, like portable devices or passive RFID tags. In that sense, a secure public-key cryptosystem with stream cipher-like design would be a breakthrough.

We first implement TCHO in software with a high-level language, and create several algorithms to compute a pseudo-random bitstream of given bias from a source of uniformly distributed random bits. We also adapt an optimized algorithm computing the output of a large LFSR, and briefly study the problem of testing the primitivity of a high-degree polynomial over $\mathbb{F}_2$. Experimental results stress out a prohibitive key generation and decryption time, in addition to limitations on the length of a plaintext, and a too high failure probability in decryption.

Then, by viewing the encryption as the communication of a codeword of some cyclic linear code over a binary symmetric channel, we generalize the construction and create derived scheme, called TCHO2. We suggest to use other codes than arbitrary LFSR ones, and study the remarkable case of block repetition codes, which allow a decryption algorithm exponentially faster, along with a sharp estimation of the error probability.

We prove the semantic security of both TCHO and TCHO2, and propose two hybrid constructions to build an IND-CCA secure system. We also introduce a new adversary model (ICCA), weaker than CCA, and study a construction for which TCHO2 is IND secure in this model. Eventually, we exhib secure asymptotic parameters, and compare to RSA.

In the ultimate chapter, we present some weaknesses of the pseudo-random generator ISAAC.

Part of this work lead to a submitted paper.

Résumé

Ce travail est basé sur un nouveau système à clé publique proposé par Finiasz et Vaudenay, possédant une construction de chiffrement de flux propice à une implantation matérielle, se démarquant ainsi des systèmes asymétriques courants nécessitant des opérations arithmétiques non triviales, banissant de ce fait l'utilisation de protocoles basé sur des schemas asymétriques dans certains environnements, comme les tags RFID passifs.

La sécurité du système repose essentiellement sur la difficulté de retrouver un multiple de poids faible d'un polynôme de haut degré et de poids quelconque sur le corps $\mathbb{F}_2$. L'algorithme de chiffrement est non-déterministe, et nécessite deux LFSR ainsi qu'une source de bits pseudo-aléatoires d'un biais donné. Le déchiffrement consiste en un algorithme de recherche exhaustive et de simples opérations d'algèbre linéaires.

Nous implémentons tout d'abord TCHO dans un langage de haut niveau ; plusieurs algorithmes sont crées pour la production de la séquence pseudo-aléatoire, et un algorithme optimisé est adapté pour le fonctionnement de LFSR longs de plusieurs milliers de bits. Experimentalement, le temps d'une génération de clé et d'un déchiffrement s'avèrent prohibitifs, de plus certaines limitations sur la taille d'un message clair, ainsi qu'une probabilité d'erreur non négligeable dans le déchiffrement et une grande expansion du chiffré, rendent le système inutilisable en pratique.

Nous proposons ensuite une variante, nommée TCHO2, réduisant exponentiellement le temps de déchiffrement, pour laquelle nous calculons précisement le taux d'erreur. La sécurité sémantique de ce nouveau système est prouvée sous certaines hypothèses, et nous proposons deux constructions hybrides garantissant l'indistinguabilité des chiffrés dans des attaques à chiffré choisi adaptives. Un nouveau modèle d'adversaire est présenté (ICCA), dans lequel nous étudions la sécurité de TCHO2 et certaines de ses variantes. Finalement, nous étudions le comportement asymptotiques des paramètres du système, et comparons avec RSA.

Enfin, le dernier chapitre présente plusieurs faiblesses observées sur le générateur pseudo-aléatoire ISAAC.

Une partie de ce travail a donné lieu à un article soumis à publication.

Note: Comme il est d'usage à l'EPFL, le rapport est rédigé en anglais, pour une meilleure accessibilité.

Notations

Notation	Name
$\mathbb{Z}$	set of integers
$\mathbb{P}$	set of prime numbers
$\mathbb{F}_q$	finite field with q elements
$\mathbb{F}_q[X]$	ring of polynomials over $\mathbb{F}_q$
$\mathbb{F}_2^n$	vector space of dimension n over $\mathbb{F}_2$
M_d	d-th Mersenne number: $2^d - 1$
$\Pr[E]$	probability of the event E, with contextual probability law
$\Pr[F\|E]$	probability of the event F conditioned to E's occurrence
$\leftarrow$	affectation
$\xleftarrow{\$}$	randomized affectation (uniform law)
$\lfloor x \rfloor$	floor: $\max\{n \| n \leq x, n \in \mathbb{Z}\}$
$\lceil x \rceil$	ceil: $\min\{n \| n \geq x, n \in \mathbb{Z}\}$
$\lfloor x \rceil$	nearest integer: $n \in \mathbb{Z}, \forall k \in \mathbb{Z}, \|x - n\| \leq \|x - k\|$
x/y	division operation
$x \| y$	division predicate: $\exists z, x \times z = y$
$\mathsf{gcd}(x, y)$	greatest common divisor of x and y
e	the transcendental number: $e = \sum_{n \geq 0} \frac{1}{n!} \approx 2.7182818$
$\log x$	binary logarithm: $\log_2 x$

$\ln x$	natural logarithm: $\log_e x$
$n!$	factorial: $\prod_{k=2}^{n} k$
$\binom{n}{k}$	binomial coefficient: $\frac{n!}{k!(n-k)!}$
$\mathcal{P}oly\,(n)$	some polynomial function in n
$\mathcal{M}^\dagger$	transpose of the matrix $\mathcal{M}$
$\deg(P)$	degree of the polynomial P
$\mathrm{ord}(P)$	order of the polynomial P: $\min\{k, X^k \equiv 1 \mod P\}$
$\mathrm{w}_p(P)$	weight: number of non-zero coefficients of the polynomial P
$x\|\|y$	concatenation of the bitstrings x and y
$\mathrm{w}_h(x)$	Hamming weight: number of non-zero bits in the bitstring x
$x \ll k$	bitstring x left-shifted of k bits "à la C" (neither circular nor expanding)
$\mathcal{L}_P$	LFSR with feedback polynomial P
$\mathcal{L}_P(x)$	*idem*, but initialized with the bitstring x
$\mathcal{S}_{\mathcal{L}_P(x)}$	bitstream generated by $\mathcal{L}_P(x)$
$\mathcal{S}_\gamma$	random bitstream with bias γ
$\mathcal{S}^\ell$	bitstream $\mathcal{S}$ truncated to its first ℓ bits
$P \otimes \mathcal{S}$	product of a polynomial and a bitstream
$\mathcal{O}\,(g(n))$	asymptotic upper bound: $f(n) = \mathcal{O}\,(g(n)) \iff \exists c > 0, \lvert f(n)\rvert \leq c\lvert g(n)\rvert$
$\Omega(g(n))$	asymptotic lower bound: $f(n) = \mathcal{O}\,(g(n)) \iff \exists c > 0, \lvert f(n)\rvert \geq c\lvert g(n)\rvert$
$o\,(g(n))$	asymptotically negligible: $f(n) = o\,(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Contents

Chapter 1

Preliminaries

This chapter introduces the background knowledge required to understand the developments following. The reader familiar with cryptology may skip most of the chapter.

1.1 Terminology

The logarithm in base 2 is denoted log, and the natural logarithm ln. The floor and ceiling of a real number r are respectively denoted $\lfloor r \rfloor$ and $\lceil r \rceil$, while the nearest integer is denoted $\lfloor r \rceil$.

An element of $\mathbb{F}_2$ is called a *bit* hereafter. An element of $\mathbb{F}_2^n$ is called a *bitstring*, where n may be finite or infinite. Its *length* $|x|$ is its number of bits. Its *Hamming weight* $\mathrm{w}_h(x)$, or simply *weight*, is its number of ones. The *Hamming distance* between two bitstrings x and y of equal length is the number of positions where x and y differ. The concatenation of x and y is $x||y$. The sum over $\mathbb{F}_2$ is denoted $+$, and the product $\cdot$ or $\times$. A bitstring x can be written $(x_1, x_2, \ldots, x_n)$, and $(0, \ldots, 0)$ can simply be denoted 0. The sum of two bitstrings of equal length returns a bitstring, and is defined as a sum component by component. It is symbolized as the usual addition by the sign $+$, we will sometimes use the neologism "to xor" to denote this operation. A *bitstream* is a bitstring of potentially infinite length produced by some device or bit source, and shall be denoted by the symbol $\mathcal{S}$ with contextual subscript. The symbol $\mathcal{S}^\ell$ refers to the bitstream $\mathcal{S}$ truncated to its first ℓ bits.

Elements of the ring $\mathbb{F}_2[X]$ are simply called *polynomials* hereafter. To lighten notations, a polynomial $P(X)$ is written P. The degree of a polynomial P is denoted $\deg(P)$, and its *weight* $\mathrm{w}_p(P)$ is its number of non-zero coefficients.

If we speak about *random* bits, or random sequence, *etc.*, it is either uniform or non-uniform randomness, and specified only where the meaning can be ambiguous. When no probability ditribution or space is specified,

randomly chosen means randomly chosen among all the objects of that kind, with respect to a uniform probability law. We may simply call *uniform bits* a sequence of uniformly distributed random bits, and *biased bits* a sequence of random bits with a certain bias.

The *statistical distance* between two probability ensembles D_1 and D_2 over $\{0,1\}^n$ is defined as

$$\mathcal{D} = \sum_{x \in \{0,1\}^n} |\Pr_{D_1}[x] - \Pr_{D_2}[x]|.$$

We shall use the acronyms CCA, CPA, IND, NM, and OW, respectively standing for the usual notions of adaptive chosen ciphertext attack, chosen plaintext attack, indistinguishability, non-malleability, and one-wayness. Corresponding definitions are recalled in a further section.

Finally, we introduce the natural values c_{easy} and c_{hard}, chosen such that an algorithm of time complexity below $2^{c_{\mathsf{easy}}}$ is considered as feasible, but intractable over $2^{c_{\mathsf{hard}}}$ (choosing $c_{\mathsf{easy}} = 40$ and $c_{\mathsf{hard}} = 80$ seems reasonable today).

1.2 Information and coding

We recall that the *length* n of a code C is the fixed number of symbols of a codeword, while the *distance* d of a code (or minimum distance) is the minimal Hamming distance between two codewords. We will only consider binary codes, *i.e.*, where the alphabet is $\{0,1\}$.

Introduced in 1948 by Claude E. Shannon [Sha48], information theory is strongly related to coding and decoding problems, some of its results are essential in the security of TCHO. In Shannon's theory, any information can be coded as a sequence of bits, so as to be transmitted from an *transmitter* (encoder) to a receiver (decoder) over a *communication channel*, which may be *noised*. We consider the model of the binary symmetric channel: each bit sent is modified with a given probability, unchanged otherwise, and no bit is added nor deleted. A random source can be defined by its *bias*:

Definition 1. *A random source of bits with bias* $-1 \leq \gamma \leq 1$ *produces a zero with probability* $p_\gamma = (\gamma + 1)/2$ *(and a one with probability* $1 - p_\gamma$*).*

That is, γ is equal to the difference between the probability to output a zero and the probability to ouput a one. We can limit us to the case of positive biases without loss of generality.

Definition 2. *The amount of randomness, or* information entropy*, of a random bitstring of length* ℓ *with bias* γ *is*

$$\ell \cdot H(p_\gamma)$$

where $H(p_\gamma)$ is the information entropy function:

$$H(p_\gamma) = -p_\gamma \log p_\gamma - (1 - p_\gamma) \log(1 - p_\gamma)$$

It thus captures the concept of *information* contained in a random bit-string, by measuring its level of uncertainty [1].

Definition 3. *The* rate *of a code of fixed length n and m words is the value*

$$R = \frac{\log m}{n}$$

Clearly, $R \leq 1$ (we cannot have more than 2^n distinct words in a code of length n). Hence a code reaches $R = 1$ when no redundancy has been added in the code, in this case no error can be detected.

Definition 4. *The* capacity *of a binary symmetric channel noised with bias γ is the value*

$$\mathcal{C}_\gamma = 1 + p_\gamma \log p_\gamma + (1 - p_\gamma) \log(1 - p_\gamma),$$

Informally, the channel capacity, is the amount of discrete information that can be reliably transmitted over a channel.

This fundamental theorem states a bound on the ability to decode on a noisy channel (see [Sha48] for the proof):

Theorem 1 (Shannon, informal)**.** *Let us be given a channel of capacity $\mathcal{C}$, with information transmitted at a rate R. There exists a way to decode with an arbitrary small error probability if and only if $R < C$.*

We now define a broadly used family of codes.

Definition 5. *A* linear code *of length n is a subspace of $\mathbb{F}_2^n$. The dimension of this subspace is called the* dimension *of the code, and usually denoted k. If the code has distance d, it is called a (n, k, d)* linear code*.*

As a consequence, any linear code has a $n \times k$ *generator matrix* G of full rank, and any matrix row equivalent to G also generates the code.

Definition 6. *A linear code C of length n is* cyclic *if, for any $c = (c_1, \ldots, c_n) \in \{0, 1\}^n$,*

$$c \in C \Rightarrow (c_n, c_1, \ldots, c_{n-1}) \in C.$$

We now give some results on the ability to detect and correct errors; by considering the spheres centered on each codeword (*i.e.* all the words at a given distance from a given codeword), the following theorem is quite intuitive:

[1] The story goes that Shannon did not know how to call this measure, so he asked Von Neumann, who said "You should call it entropy (...) [since] no one knows what entropy really is, so in a debate you will always have the advantage", see [TM71] for more details.

Theorem 2. *A code of distance d can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors.*

Proof. The spheres of radius $\lfloor \frac{d-1}{2} \rfloor$ centered on each codeword do not overlap, thus any codeword with at most this amount of errors belongs to a single sphere, and decoding only consists in choosing the center of this sphere. □

The following bounds on linear codes are given without proof.

Theorem 3 (Hamming)**.** *If C is a (n, k, d) linear code, with $d = 2t + 1$ or $2t + 2$, then*

$$|C| \sum_{i=0}^{t} \binom{n}{i} \leq 2^n$$

Theorem 4 (Singleton)**.** *For any (n, k, d) linear code, $d \leq n - k + 1$.*

Those definitions are the minimal requirements for the understanding of the coding related parts of this report, for further theory one can refer to the reference [Rom92], [HWL+91] for a concise introduction to the subject, or even [MS77] for an intermediate approach, also dealing with pure information theory.

1.3 Stream ciphers

1.3.1 Generalities

Stream ciphers used to be symmetric ciphers, producing a bitstream (called the *keystream*) defined by the secret key, combined with the message to build the ciphertext, and can thus be depicted as *keystream generators*, devices producing a random looking bitstream from a certain key. The combination is the most often defined as a simple XOR, but more general definitions exist. Stream ciphers can often be seen as *pseudo-random generators*. The stream cipher paragon is the *Vernam cipher* [Ver26], proved unconditionnally secure by Shannon in 1949, under the condition that each random sequence is used only once, introducing the notion of *perfect secrecy*. One can wonder why we do not simply use pseudo-random generators as stream ciphers; the main difference is that stream ciphers' bitstreams must be defined by a unique key, belonging to a large enough key space, satisfy several statistical properties to be declared cryptographically secure, and reach good hardware and/or software performances so as to be effectively used.

Stream ciphers can be either synchronous or self-synchronous: in the first case, the keystream only depends on the key, whilst in the second it also depends on the previous encrypted bits (for example, the CFB operation mode of block ciphers). Some famous stream ciphers are A5/1 (used in GSM encryption), E0 (used in Bluetooth protocol), RC4 (used in SSL and WEP), SEAL, SOBER, SNOW, Phelix, *etc.*

1.3.2 The linear feedback shift register

The linear feedback shift register (LFSR) is a structure widely used in the design of stream ciphers, either in its original form, or under variants like the self-shrinking generator [MS94] or the Galois LFSR. Here, after short preliminaries, we introduce our formalism and state some remarkable properties of the LFSR and its outputs.

On polynomials

We call *binary polynomial* an element of the ring $\mathbb{F}_2[X]$. Each binary polynomial can be written under the normal form

$$\sum_{i=0}^{\infty} c_i X^i,$$

where the number of non-zero coefficients is finite. We will only deal with binary polynomials, and simply call them polynomials.

These two routine definitions are essential for the following developments:

Definition 7. *The* order *of $P \in \mathbb{F}_2[X]$ is the smallest integer $k \geq 1$ such that $X^k \equiv 1 \mod P$.*

Definition 8. *$P \in \mathbb{F}_2[X]$ is said to be* primitive *if its order is $2^{\deg(P)} - 1$ (the maximal possible order for this degree).*

More precisely, an irreducible polynomial of degree d is said to be primitive if its root in the splitting field $\mathbb{F}_{2^d}$ is a generator of the multiplicative group $\mathbb{F}_{2^d}^\star$.

The next proposition is just the application of a famous theorem of Lagrange:

Proposition 1. *The order of any irreducible binary polynomial P of degree d divides $2^d - 1$.*

Corollary 1. *If $2^d - 1$ is prime, then any irreducible binary polynomial of degree d is primitive.*

Definition

A binary LFSR $\mathcal{L}_P$ of length n is a device aimed at producing a bitstream, composed of a register of n bits $(s_i, \ldots, s_{i+n-1})$, and a linear feedback function, characterizing the update the register. We only consider LFSR's where the register values are elements of $\mathbb{F}_2$, but one can also build LFSR's on $\mathbb{F}_q$, for some $q = p^n$, $p \in \mathbb{P}$. The register content is usually called the *state* of the LFSR, and $(s_0, \ldots, s_{n-1})$ the *initial state*, which entirely determines the bitstream produced. The feedback function is defined by a polynomial

$P = \sum_{i=0}^{\infty} p_i X^i$ of degree n, called the *feedback polynomial*, of degree equal to the LFSR length. The offsets where P has non-zero coefficients are often called *taps*. The update of the state is described by the following linear operation[2]:

$$\begin{pmatrix} s_i \\ \vdots \\ \vdots \\ \vdots \\ s_{i+n-1} \end{pmatrix}^{\dagger} \begin{pmatrix} 0 & 0 & \dots & 0 & p_1 \\ 1 & 0 & \dots & \vdots & p_2 \\ 0 & 1 & & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & p_{n-1} \\ 0 & \dots & 0 & 1 & p_n \end{pmatrix} = \begin{pmatrix} s_i + 1 \\ \vdots \\ \vdots \\ \vdots \\ s_{i+n} \end{pmatrix}^{\dagger},$$

and so the bitstream produced with an initial state $x \in \mathbb{F}_2^n$ is

$$\mathcal{S}_{\mathcal{L}_P(x)} = (s_0, \dots, s_i, \dots).$$

A LFSR is a weak source of random information; bits are strongly correlated, and the sequence is condemned to be ultimately periodic, since the number of distinct states is finite: only $2^n - 1$ (the all-zero state is discarded). A LFSR is called *optimal* when its period is maximal, *i.e.* equal to the number of possible non-zero states. By representing the keystream as a generating function, some routine calculus leads us to the following proposition (one will refer to any good book or lecture notes for the proof):

Proposition 2. *The period of a LFSR is equal to the order of its feedback polynomial.*

Corollary 2. *A LFSR of size n achieves its maximal period $2^n - 1$ if and only if its feedback polynomial is primitive.*

Thus for any non-zero initial state, if the feedback polynomial is primive, then all the non-zero states will appear in a period.

Properties of the bitstream

Definition 9. *The product of a binary polynomial $K = \sum_{i=0}^{\infty} k_i X^i$ of degree d and a bitstream $\mathcal{S}^{d+N} = (s_0, \dots, s_{d+N-1})$ is defined as*

$$K \otimes \mathcal{S}^{d+N} = (s'_0, \dots, s'_{N-1})$$

with

$$s'_i = s_i k_0 + s_{i+1} k_1 + \cdots + s_{i+d} k_d.$$

[2]One may also find in literature different formalisms where the polynomial is reversed, *i.e.*, where it is the reciprocal of the characteristic polynomial of the recurrence, but both are canonical, and equivalent.

The operator thus defined is distributive over the bitstring sum and the polynomial sum, it verifies

$$(PQ) \otimes \mathcal{S} = P \otimes (Q \otimes \mathcal{S}) \text{ and } P \otimes \mathcal{S}_{\mathcal{L}_P(x)} = 0$$

for all $P, Q \in \mathbb{F}_2[X]$, any bitstream $\mathcal{S}$, and any bitstring x of length $\deg(P)$. As a consequence we have:

Fact 1. *$\forall P, Q \in \mathbb{F}_2[X]$ with non-zero constant term, $\forall R \in \mathbb{F}_2[X]$, $\forall x \in \mathbb{F}_2^{\deg(P)}$, $\forall y \in \mathbb{F}_2^{\deg(Q)}, \forall \ell > \deg(QR)$,*

$$(QR) \otimes (\mathcal{S}^{\ell}_{\mathcal{L}_P(x)} + \mathcal{S}^{\ell}_{\mathcal{L}_Q(y)}) = (QR) \otimes \mathcal{S}_{\mathcal{L}^{\ell}_P(x)}.$$

This result will be used to "delete" a LFSR bitstream, in the decryption procedure of TCHO. We now state a bridge with coding theory:

Fact 2. *Let P be a polynomial of degree d_P. The set $\{\mathcal{S}^{\ell}_{\mathcal{L}_P(x)}, x \in \mathbb{F}_2^d\}$ is a cyclic linear code of length ℓ and dimension at most 2^d.*

Security of LFSR-based stream ciphers

In practice, one never uses the textbook LFSR as a stream cipher, but one or several LFSR's combined with non-linear operations, such as permutations, boolean functions with high algebraic degree, or more exotic constructions. Examples classical design techniques are the non-linear combination generator, the non-linear filter generator, or the clock-controlled generator.

Basically, the goal of an attacker is to recover all or part of the initial state of the LFSR (or any information related), from a source of information depending on the security model considered. Generally, an attack is performed when a few bits of the keystream are known, successive or not. Note that an attacker gains nothing in chosen ciphertext attack compared to a chosen plaintext attack or a known plaintext attack, since the information obtained on the secret (the keystream bits) is exactly the same (this stands only when the combination can be inversed, *e.g.* if it is a simple XOR). The problem of finding the minimal polynomial producing a given LFSR stream has also been investigated, and lead to the well-known Berlekamp-Massey algorithm [Ber68].

Brute force attacks Assuming that the construction does not allow us to easily recompute the initial state of a LFSR from all or part of the keystream, the first naive attack to retreive this secret is the try-and-test approach, the so-called *exhaustive search*. A secure stream cipher is often defined as one where it is the best possible attack, the Grail of cryptographers. The average time complexity of this attack is clearly in $\Omega(2^{n-1})$, where n is the number of secret bits. As usual, time-memory trade-off can reduce this cost (*cf.* dictionary and codebook attacks).

Correlation attacks This famous attack was discovered by Siegenthaler in 1985 [Sie86], then improved by Meier and Staffelbach [MS88] who presented it as a decoding problem; the general idea is to find a statistically biased distribution between the keystream and a bitstream produced by another source, typically a LFSR. It can lead for example to reduce the attack to the noisy decoding of the code spanned by another LFSR. Several decoding algorithms have been proposed; maximum likelihood, Gallagher's iterative decoding of low-density parity-check codes, turbo codes, *etc.* In particular, correlation attacks were used to attack the widely used E0 [LV04a, Ekd03, HN99], and even the RC5 block cipher [MNT02].

Other attacks Below, for historic purposes, we give a non-exhaustive list of known attacks on stream ciphers (LFSR-based or not):

- key reuse (medieval),
- correlation (Siegenthaler, 1984),
- guess-and-determine (Günter, 1988),
- resynchronization (Daemen *et al.*, 1993),
- time-memory tradeoffs (Babbage, 1995),
- backtracking (Golic, 1997),
- algebraic (Shamir *et al.*, 1999),
- side channel (Kocher *et al.*, 1999),
- binary decision diagrams (Krause, 2002).

1.4 Public-key cryptography

Public-key cryptography was discovered by Diffie and Hellman [DH76] in 1976[3]. Since, dozens of cryptosystems appeared, based on hard problems like integer factorization, discrete logarithm, lattice reduction, knapsacks, *etc.*, in various algebraic structures. A public-key (or asymmetric) cryptosystem consists of an encryption procedure, requiring an element pk, along with the associated decryption procedure which requires an element sk. The element pk is made publicly available, and called the *public key*, while sk is kept secret, and called the *private key*. The system must satisfy the property that it is computationally infeasible to recover sk from pk. More formally, we give the following definition.

[3]Ellis [Ell70] discovered it independently in 1970, but his works were classified by a British government agency until 1997.

Definition 10. *A public-key cryptosystem is defined by three sets and three algorithms. The sets are:*

- $\mathcal{M}$*, the plaintexts space, finite or infinite.*
- $\mathcal{C}$*, the ciphertexts space, finite only if* $\mathcal{M}$ *is finite.*
- $\mathcal{R}$*, the random coins space, finite, and non-empty only if encryption is probabilistic.*

The three algorithms are:

- *The* key generation algorithm $\mathcal{G}$*, which outputs a pair* $(\mathsf{pk}, \mathsf{sk})$ *of matching public and private keys, on input* 1^k*, where* k *is the security parameter.*
- *The* encryption algorithm $\mathcal{E}$*, which, given a plaintext* $m \in \mathcal{M}$ *and a public key* pk*, outputs a ciphertext* $c \in \mathcal{C}$ *of* m*. This algorithm may be probabilistic (involving random coins).*
- *The* decryption algorithm $\mathcal{D}$*, which, given a ciphertext* $c \in \mathcal{C}$ *and a private key* sk*, returns the matching plaintext* $m \in \mathcal{M}$*, or* $\perp$ *if the given ciphertext is not valid.*

Asymmetric systems are seldom used alone, but as part of an hybrid encryption scheme, to encrypt the secret key of a symmetric scheme, which encrypts the message. This technique is often refered as a key encapsulation mechanism and data encapsulation mechanism ("KEM/DEM"). We will meet such constructions later.

Public key cryptosystems are also closely related to the notions of one-way and trapdoor functions, but it comes out of the scope of this report (see for example [BHSV98, Yao82]).

1.5 Security definitions

An adversarial model is the statement of what an adversary (*i.e* one or several probabilistic algorithms querying oracles) can and cannot do when attacking the encryption scheme, so as to study the security of the system. For public-key schemes, anyone can encrypt any message, so the basic attack is the chosen plaintext attack (CPA). The number of queries is limited to a polynomially bounded number. For symmetric schemes, CPA attacks are modeled using encryption oracle, whilst in the most basic attack the adversary only has a ciphertext of an unknown message. The best security level is achieved in the following model:

Definition 11. *An adversary is called an* adaptive chosen ciphertext *(CCA) adversary if she can query the decryption oracle whenever she wants, to decrypt any ciphertext except the given challenge(s). The number of queries must be polynomially bounded. If the ciphertext given to the oracle is not a valid one, the oracle returns $\perp$, and the attack continues.*

In literature, this model is sometimes denoted CCA2, and CCA is standing for non-adaptive adversaries, where queries to the decryption oracle do not depends on the challenge.

Now we review fundamental security notions: one-wayness, indistinguishability, real-or-random security, non-malleability, and semantic security. We recall that in CCA model, the adversary cannot query the decryption oracle with the ciphertext computed by the challenger.

Definition 12 (OW security)**.** *Let $\mathcal{A}^{ow} = (\mathcal{A}_1^{ow}, \mathcal{A}_2^{ow})$ be an adversary involved in the following game:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{G}(1^k)$*: a key pair is generated.*
2. $\sigma \leftarrow \mathcal{A}_1^{ow}(\mathsf{pk})$*: the adversary queries oracle(s) and return a state.*
3. $m \xleftarrow{\$} \mathcal{M}$*: a plaintext is randomly picked by the challenger.*
4. $c = \mathcal{E}(\mathsf{pk}, m)$*: the challenger encrypts m and sends c to the adversary.*
5. $\tilde{m} \leftarrow \mathcal{A}_2^{ow}(\sigma, c)$*:*

The advantage of an adversary $\mathcal{A}^{ow}$ against one-wayness is

$$\mathsf{Adv}^{ow} = \Pr[m = \tilde{m}].$$

We say that a cryptosystem is (t, ε)-OW secure when any adversary running in time less than t gets an advantage less than ε.

Definition 13 (IND security)**.** *Let $\mathcal{A}^{ind} = (\mathcal{A}_1^{ind}, \mathcal{A}_2^{ind})$ be an adversary involved in the following game:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{G}(1^k)$*: a key pair is generated, and* pk *sent to $\mathcal{A}_1^{ind}$.*
2. $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{ind}(\mathsf{pk})$*: the adversary returns a pair of plaintexts of equal length, and a state σ.*
3. $b \xleftarrow{\$} \{0, 1\}$*: a random bit is picked by the challenger.*
4. $c \leftarrow \mathcal{E}(\mathsf{pk}, m_b)$*: the challenger encrypts m_b, and sends c to $\mathcal{A}_2^{ind}$.*
5. $\tilde{b} \leftarrow \mathcal{A}_2^{ind}(m_0, m_1, \sigma, c)$*: the adversary guesses the message which was encrypted.*

The advantage of an adversary $\mathcal{A}^{ind}$ against indistinguishability is

$$\mathsf{Adv}^{ind} = 2\Pr[b = \tilde{b}] - 1.$$

We say that a cryptosystem is (t, ε)-IND secure when any adversary running in time less than t gets an advantage less than ε.

Definition 14 (ROR security)**.** *Let $\mathcal{A}^{ror} = (\mathcal{A}_1^{ror}, \mathcal{A}_2^{ror})$ be an adversary involved in the following game:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{G}(1^k)$*: a key pair is generated, and* pk *sent to* $\mathcal{A}_1^{ror}$*.*
2. $m_0 \xleftarrow{\$} \mathcal{M}, b \xleftarrow{\$} \{0,1\}$*: the challenger picks a random plaintext and a value b.*
3. $(m_1, \sigma) \leftarrow \mathcal{A}_1^{ror}(\mathsf{pk})$*: the adversary chooses a plaintext, sent to the challenger.*
4. $c \leftarrow \mathcal{E}(\mathsf{pk}, m_b)$*: the challenger encrypt m_b, and sends it to $\mathcal{A}_2^{ror}$.*
5. $\tilde{b} \leftarrow \mathcal{A}_2^{ror}(m_1, \sigma, c)$*: the adversary guesses whether her message or another one was encrypted.*

The advantage of an adversary $\mathcal{A}^{ror}$ in a real-or-random game is

$$\mathsf{Adv}^{ror} = 2\Pr[b = \tilde{b}] - 1.$$

We say that a cryptosystem is (t, ε)-ROR secure when any adversary running in time less than t gets an advantage less than ε.

Definition 15 (NM security)**.** *Let $\mathcal{A}^{nm} = (\mathcal{A}_1^{nm}, \mathcal{A}_2^{nm})$ be an adversary involved in the following game:*

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{G}(1^k)$*: a key pair is generated, and* pk *sent to* $\mathcal{A}_1^{nm}$*.*
2. $(M, \sigma) \leftarrow \mathcal{A}_1^{nm}(\mathsf{pk})$*: a distribution of plaintexts M and a state σ are returned by $\mathcal{A}_1^{nm}$.*
3. $(m, \tilde{m}) \xleftarrow{\$} M$*: the challenger picks two independent random plaintexts according to the distribution M.*
4. $c \leftarrow \mathcal{E}(\mathsf{pk}, m)$*: m is encrypted and sent to $\mathcal{A}_2^{nm}$.*
5. $(R, y) \leftarrow \mathcal{A}_2^{nm}(m, \sigma, c)$*: the adversary computes a binary relation R and a ciphertext y.*
6. $x \leftarrow \mathcal{D}(\mathsf{sk}, y)$*.*

The advantage of an adversary against non-malleability is

$$\mathsf{Adv}^{nm} = \Pr\left[y \neq c \wedge x \neq \perp \wedge R(x, m)\right] - \Pr\left[y \neq c \wedge x \neq \perp \wedge R(x, \tilde{m})\right].$$

We say that a cryptosystem is (t, ε)-NM secure when any adversary running in time less than t gets an advantage less than ε.

This last definition models the informal notion of non-malleability: an adversary cannot compute a ciphertext *meaningfully related* to the message matching a given distinct ciphertext. And so a malleable cryptosystem does not guarantee integrity of the ciphertexts, but may ensure privacy.

A cryptosystem is told to be X-Y secure if it guarantees X security in the attack model Y. In our context, we simply call X-Y secure a system which is (t, ε)-X-Y secure, with $t \geq 2^{c_{\mathsf{hard}}}$, and ε negligible (*i.e.* $\varepsilon \leq 2^{-c_{\mathsf{hard}}}$), where X can be either IND, OW or ROR, and Y can be either CPA, or CCA.

The two following results are proved in [BDPR98]:

Proposition 3. *NM-CPA security implies IND-CPA security.*

Proposition 4. *NM-CCA security is equivalent to IND-CCA security.*

More generally, NM-Y $\Rightarrow$ IND-Y $\Rightarrow$ OW-Y, for all adversarial model Y, and X, X-CCA security implies X-CPA security, for all security notion X. The next proposition is proved in [BDJR97]:

Proposition 5. *ROR-CPA security is equivalent to IND-CPA security.*

The important notion of *semantic security* introduced by Goldwasser and Micali [GM82] is equivalent [GM84] to IND-CPA security; it guarantees that the ciphertext reveals no more information about the plaintext to a polynomially bounded adversary. Note that semantic security implies OW-CPA security, the weakest level of security.

To summarize, we obtain the followin relationship:

$$\begin{array}{rcl} & \mathsf{NM\text{-}CPA} & \Leftarrow \ \mathsf{NM\text{-}CCA} \\ & \Downarrow & \Updownarrow \\ \mathsf{Semantic} \Leftrightarrow & \mathsf{IND\text{-}CPA} & \Leftarrow \ \mathsf{IND\text{-}CCA} \\ & \Downarrow & \Downarrow \\ & \mathsf{OW\text{-}CPA} & \Leftarrow \ \mathsf{OW\text{-}CCA} \end{array}$$

Chapter 2

The TCHO scheme

A trapdoor stream cipher sounds like a premiere in cryptography, but it is not exactly one: in 1984 Blum and Goldwasser [BG85] used the Blum-Blum-Shub [BBS86] pseudo-random generator to build a probabilistic public-key stream cipher based on the hardness of factoring a RSA modulus, and on the security of the generator (see Appendix B for details and discussion). However it is more or less as computationally expensive as RSA, not well fitted for hardware as many streams ciphers do, and not really a trapdoor stream cipher in the strict sense. The idea of putting a trapdoor in a LFSR-based stream cipher has been brought by Camion, Mihaljevic and Imai three years ago [CMI03], but no explicit cryptosystem followed. As a response, the system TCHO[1] aims at providing a secure trapdoor stream cipher hardware-friendly, and being the first *real* asymmetric stream cipher. Encryption is probabilistic, and can be described as the transmission of a codeword over a noisy channel, as depicted in Figure 2.1: one small LFSR encodes the message, while a large one randomly initialized, along with a source of biased random bits, produces the noise. A ciphertext is the XOR of the three bitstreams. The private key is used to "cancel" the bitstream of the second LFSR, thereby reducing the noise over the coded message, so as to be able to decode the cyclic linear code spanned by the small LFSR.

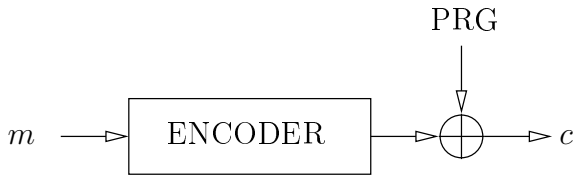

Figure 2.1: TCHO encryption scheme.

[1] See `http://www.tcho.fr` for the origins of this name.

2.1 Computational problems

The security of TCHO relies on the hardness of two distinct computational problems; one dealing with sparse multiples of primitive polynomials over $\mathbb{F}_2$, and a famous one related to some decoding problems, strongly linked with the stream ciphers cryptanalysis field. The two problems are then merged into a single one. In this section we introduce these problems, and state hardness assumptions in terms of their parameters, regarding to the known attacks of these problems.

2.1.1 Finding a sparse multiple of a high-degree polynomial

This problem formalizes the key recovery problem in TCHO:

Low Weight Polynomial Multiple (lwpm)
Parameters: Three naturals w, d and d_P, such that $0 < d_P < d$ and $w < d$.
Instance: $P \in \mathbb{F}_2[X]$ of degree d_P.
Question: Find a multiple K of P of degree less than d and weight less than w.

Unlike integers, efficient methods are known to factorize a polynomial over a finite field (Berlekamp's generic deterministic algorithm runs in polynomial time in the input's degree, whereas the cheapest method known for integers is the super-polynomial GNFS), but finding a multiple of degree and weight below certain bounds can be hard. This problem, or its variants, has been important in LFSR cryptanalysis since some attacks are possible only when the feedback polynomial or one of its multiple is sparse [MS88, CT00]. A few works [GM01, MGV05, Jam00] study the distribution of multiples of a given weight, but consider the problem of finding a sparse multiple without the constraint on the degree. If d is greater than to the order n of P, a trivial solution is the polynomial $X^n + 1$, choosing primitive polynomials would avoid this concern.

We can compute the average number $\mathcal{N}_{\mathsf{sol}}$ of solutions of a lwpm instance. The probability that exists a multiple of P with degree d and weight w is heuristically

$$2^{d-d_P} \frac{\binom{d}{w-1}}{2^d} = 2^{-d_P} \binom{d}{w-1},$$

and so

$$\mathcal{N}_{\mathsf{sol}} \approx 2^{-d_P} \sum_{i=1}^{d} \sum_{j<d, j<w} \binom{i}{j}.$$

In [GM01], an exact enumeration formula is given for the number of multiples of weight v (of unbounded degree, with constant term 1) of any primitive polynomial of given degree. Although this expression is useless here, since we need a multiple of a specific degree, it gives an idea of the problem hardness.

Example 1. *A primitive polynomial of degree* 10 *has about* 10^{18} *multiples of weight* 10 *with constant term* 1*, but only* 339 *of weight* 3*.*

We now present strategies to solve LWPM. The following are suggested (refer to [FV06] for more details):

1. Birthday paradox: memory $\mathcal{O}\left(2^{d_P/2}\right)$, time $\mathcal{O}\left(d_P 2^{d_P/2}\right)$ for a single solution, and $\mathcal{O}\left(L \log L\right)$ for all solutions with $L = \binom{d}{(w-1)/2}$.
2. Generalized birthday paradox [Wag02]: time $\mathcal{O}\left(2^{a+\frac{d_P}{a+1}}\right)$, if there exists an $a \geq 2$ such that $\binom{d}{(w-1)/2^a} \geq 2^{d_P/(a+1)}$.
3. Syndrome decoding [CC98, LB88]: time $\mathcal{O}\left(\mathcal{P}oly\left(d\right)\left(\frac{d}{d_P}\right)^{w-1}\right)$.
4. Exhaustive search: time $\mathcal{O}\left(\mathcal{P}oly\left(d\right)2^{d-d_P}\right)$ for all solutions.

An analysis of these strategies leads to a first assumption:

Assumption 1 ([FV06])**.** *When* P *is randomly chosen among the primitive factors of an unknown sparse polynomial, if* $\binom{d}{w-1} \leq 2^{d_P}$ *and* $w \log \frac{d}{d_P} \geq c_{\mathsf{hard}}$*, then* LWPM *is hard, on average.*

2.1.2 Decoding a LFSR code

Our second problem goes as follows:

Noisy LFSR Decoding (NLD)
Parameters: $P \in \mathbb{F}_2[X]$ of degree d_P, a natural ℓ, a bias $0 \leq \gamma \leq 1$.
Instance: $y = \mathcal{S}^{\ell}_{\mathcal{L}_P(x)} + \mathcal{S}^{\ell}_{\gamma}$.
Question: Recover x.

The following strategies are suggested:

1. Information set decoding: the idea is to randomly pick d_P bits of y, and solve the linear system induced by the LFSR. To recover x, we need to pick only bits with no error. The probability of this event is $p_\gamma^{d_P}$. If $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$, it requires over $2^{c_{\mathsf{hard}}}$ iterations.
2. Maximum likelihood decoding (MLD): this brute force technique consists in trying every possible initialization, and return the one minimizing the Hamming distance between y and the stream produced. The algorithm has a time complexity in $\mathcal{O}\left(\ell \cdot 2^{d_P}\right)$ (it can be decreased to $\mathcal{O}\left(d_P \cdot 2^{d_P}\right)$ by using a fast Walsh transform [LV04a]).

3. Iterative decoding: the idea of this approach is to find low weight multiples of P forming some parity check equations, and then decode in the low-density parity-check code associated (see [CT00]). For $d_P \geq 2c_{\mathsf{hard}}$, decoding is impossible.

A second assumption can thus be formulated:

Assumption 2 ([FV06]). *If $d_P \geq 2c_{\mathsf{hard}}$ and $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$, then* NLD *is hard.*

We can also state when the problem is solvable[2]:

Fact 3 ([FV06]). *If $d_Q \leq c_{\mathsf{easy}}$ and $\sqrt{\frac{d_Q \ln 4}{\ell - d}} \leq \gamma^w$, then* NLD *can efficiently be solved.*

The link with the correlation attacks becomes obvious: $\mathcal{S}^\ell_{\mathcal{L}_P} + \mathcal{S}^\ell_\gamma$ is correlated with $\mathcal{S}^\ell_{\mathcal{L}_P}$, with correlation $1 - p_\gamma$.

2.1.3 The hidden correlation problem

We now merge LWPM and NLD into a single problem.

HIDDEN CORRELATION (HC)
Parameters: Two coprime polynomials P and Q, of degree respectively d_P and d_Q, a natural ℓ, and a $0 \leq \gamma \leq 1$.
Instance: $y = \mathcal{S}^\ell_{\mathcal{L}_Q(x)} + \mathcal{S}^\ell_{\mathcal{L}_P(r)} + \mathcal{S}^\ell_\gamma$, with r an unknown random bitstream of length $\deg(P)$.
Question: Recover x.

Coprime polynomials are required so that the decoding is not ambiguous. A HC instance can be reduced to an NLD instance: if we multiply y by a multiple K of P, of degree d, by the we get the stream

$$z = K \otimes (\mathcal{S}^\ell_{\mathcal{L}_Q(x)} + \mathcal{S}^\ell_{\mathcal{L}_P(r)} + \mathcal{S}^\ell_\gamma) = K \otimes (\mathcal{S}^\ell_{\mathcal{L}^\ell_Q(x)} + \mathcal{S}^\ell_\gamma).$$

This bitstream is thus of length $\ell - d$. By linearity, we obtain a stream produced by $\mathcal{L}_Q$ with initial state x', with noise of bias γ^w, since each bit a sum of w bits noised with bias γ:

$$\mathcal{S}_{\mathcal{L}_Q(x')} + \mathcal{S}^\ell_{\gamma^w}.$$

Note that the noisy bits with bias γ^w are correlated, depending on the offsets of the non-zero coefficients of K. Experiments show that $K \otimes \mathcal{S}^\ell_\gamma$ behaves mostly like $\mathcal{S}^{\ell-d}_{\gamma^w}$.

[2]The lower bound on γ follows from an approximation of Shannon's bound obtained using $\mathcal{C}(\gamma) \approx \gamma^2 / \ln 4$.

The matrix $\mathcal{M}_f$ of the linear application transforming the real initial state x of $\mathcal{L}_Q$ to the new initialization x' can be calculated, using basic linear algebra. Let $\mathcal{M}_Q$ be the generating matrix of $\mathcal{L}_Q$ (as presented in Section 1.3.2), and $(k_i)_{i=0,\dots,\infty}$ the coefficients of K, then we have

$$\mathcal{M}_f = \sum_{i=0}^{d} k_i (\mathcal{M}_Q)^{d-i} \tag{2.1}$$

where the sum operation is the usual matrix addition. Therefore to retrieve x, given the initial state of the $\mathcal{L}_Q$ in z, it suffices to inverse this matrix. It can be done in time in $\mathcal{O}\left(d_Q^3\right)$ by Gauss-Jordan elimination. To summarize, we reduced an HC instance to one of NLD with parameters $(\ell - d, Q, \gamma^w)$.

Other strategies than this reduction are proposed to solve HC:

- Consider $\mathcal{L}_P$ and $\mathcal{L}_Q$ as a single LFSR, recover its initialization (*i.e.* solve an instance of NLD with parameters $(\ell, P \times Q, \gamma)$), and deduce those of each LFSR.
- Multiply the ciphertext by Q to cancel $\mathcal{S}^{\ell}_{\mathcal{L}_Q}$ and, recover the initial state of $\mathcal{L}_P$, by the same process that described above, except that here the NLD instance has parameters $(\ell - d, P, \gamma^{d_Q})$

By Assumption 2, we cannot solve NLD for $\mathcal{S}^{\ell}_{\mathcal{L}_P} + \mathcal{S}^{\ell}_{\gamma}$ when $d_P \geq 2c_{\mathsf{hard}}$ and $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$, thus these strategies are infeasible for well chosen parameters.

A last constraint is linked to theoretical concerns; if we suppress the influence of P using a multiple K of weight w, the information I_γ one can get on $\mathcal{S}_{\mathcal{L}_Q(x)}$ is bounded:

$$I_\gamma \leq \ell \cdot \mathcal{C}_{\gamma^w}$$

It also gives us a large bound on the information one can obtain on x. Now, what if an opponent computes *all* the multiples of P of a given weight w ? There are at most $\frac{2^{c_{\mathsf{hard}}}}{\ell w}$ multiples of weigth w, and we need ones of degree lower than ℓ, then at most $\binom{\ell}{w} 2^{-d_P}$ are suitable. We deduce the total information one can get, neglecting the cost of finding such multiples:

$$\mathcal{I} = \sum_{w=2}^{\infty} \ell \cdot \mathcal{C}_{\gamma^w} \min\left(\binom{\ell}{w} 2^{-d_P}, \frac{2^{c_{\mathsf{hard}}}}{\ell w}\right).$$

We deduce the assumption of HC's hardness:

Assumption 3 ([FV06]). *When P is randomly chosen among the primitive factors of an unknown sparse polynomial, if $d_P \geq 2c_{\mathsf{hard}}$ and $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$, and $\mathcal{I} \leq 1$, then* HC *is hard, on average.*

2.2 The public-key scheme

A public key of TCHO is a high-degree primitive polynomial P, the private key associated is a polynomial K, sparse multiple of P. TCHO encrypts blocks of d_Q bits. The parameters of the system are

- $[d_{\min}, d_{\max}]$, an interval containing d_P,
- d and w, the degree and the weight of K,
- γ, the bias of the random source,
- ℓ, the length of a ciphertext of one block,
- Q, a polynomial of small degree d_Q (the length of a plaintext block).

2.2.1 Key generation

To find a key pair, one first randomly picks a polynomial of degree d and weight w, then decomposes it into irreducible factors, and looks for a primitive polynomial of degree in $[d_{\min}, d_{\max}]$ among those factors. Avoiding the cost of testing primitivity, which is discussed later, the time complexity of the key generation is in $\mathcal{O}\left(\frac{d_{\max}}{d_{\max}-d_{\min}} d^2\right)$, using the Cantor-Zassenhaus algorithm [CZ81a].

Some trick can be used for the second step: the product of all irreducible polynomials of degree dividing d is $X^{2^d} - X$, so we can compute $\mathsf{gcd}(X^{2^d} - X \bmod K, K)$; if the polynomial computed has degree lower than d, one knows that K has no factor of degree d, otherwise we factorize the polynomial to find one. Although this technique speeds up the process for a single degree, in the worst case we would have to perform it for each degree in the range (for a single iteration), that too much increases the time complexity of the algorithm induced.

2.2.2 Encryption and decryption

Let x be a plaintext, *i.e.*, an element of $\{0,1\}^{d_Q}$. A ciphertext of x is defined as

$$\mathrm{TCHO}_{\mathsf{enc}}(x, r) = \mathcal{S}^{\ell}_{\mathcal{L}_Q(x)} + \mathcal{S}^{\ell}_{\mathcal{L}_P(r_1)} + \mathcal{S}^{\ell}_{\gamma}(r_2)$$

where $r = r_1 || r_2$ is a random bitstring of sufficient length, so the encryption is clearly non-deterministic (it is necessary to guarantee the semantic security).

One inherent weakness of TCHO is its high message expansion; to suit the constraints d_Q have to be much smaller than ℓ.

Let y be a ciphertext, *i.e.*, an element of $\{0,1\}^{\ell}$. To recover the plaintext we first compute

$$K \otimes y \approx \mathcal{S}^{\ell-d}_{\mathcal{L}_Q(x')} + \mathcal{S}^{\ell-d}_{\gamma^w}$$

where x' is the image of x by some invertible linear application f. Then, we perform a maximum likelihood decoding (MLD) to recover x', and finally compute $x = f^{-1}(x')$. As stated previously, if $\mathcal{M}_Q$ is the generating matrix of $\mathcal{L}_Q$, the matrix of f is

$$\mathcal{M}_f = \sum_{i=0}^{d} k_i (\mathcal{M}_Q)^{d-i},$$

which can be inversed in time $\mathcal{O}\left(d_Q^3\right)$. Note that this matrix does not depends on the ciphertext received, thus it can be precomputed.

The cost of MLD is in $\mathcal{O}\left((\ell - d) \cdot 2^{d_Q}\right)$, and $\mathcal{O}\left(d_Q \cdot 2^{d_Q}\right)$ using a fast Walsh transform [LV04b]. The soundness of the system is not guaranteed, since $\mathcal{S}_\gamma^\ell$ can take any value of $\{0,1\}^\ell$ with non-zero probability, and so decoding may fail if the pseudo-random bitstream has a high weight. Indeed every element of $\{0,1\}^\ell$ has a non-null probability to be obtained by encrypting some plaintext, since $\mathcal{S}_\gamma^\ell$ takes all values in $\{0,1\}^\ell$ with non-null probability. Thus the ciphertext space is $\{0,1\}^\ell$. However, not all ciphertexts are decrypted successfully, and for a given key pair, the ciphertext space can be partitionned into two sets: those which are correctly decrypted (the *sound ciphertexts*), and the others (the *non-sound ciphertexts*). We do not employ the adjective *valid* since it usually stands for an object that cannot have been produced by the encryption algorithm.

Since the complexity of decoding is not linear in d_Q, we can think we had better use a small polynomial Q (*i.e.* smaller blocks), but the matter is that the ciphertext expansion factor does not linearly grow in terms of d_Q. Thus we would encounter some problems of time-memory trade-off, and the choice of a set of parameters may depend on the user's requirements in time and amount of data encrypted.

2.2.3 Parameters selection

Refering to the above-stated assumptions, we give security constraints on the parameters.

- In order to decrypt successfully, we must be able to decode a codeword of length $\ell - d$ of a random LFSR code, noised with bias γ^w, so we need
$$d_Q \leq c_{\mathsf{easy}} \text{ and } \sqrt{\frac{d_Q \ln 4}{\ell - d}} \leq \gamma^w. \tag{2.2}$$
- Message recovery is assumed to be hard if
$$\gamma \leq 2^{1 - \frac{c_{\mathsf{hard}}}{d_{\min}}} - 1 \text{ and } \mathcal{I} \leq 1. \tag{2.3}$$

- Finally, the private key K must be impossible to recover. It is assumed to be the case as soon as

$$\binom{d}{w-1} \leq 2^{d_{\min}} \text{ and } w \log \frac{d}{d_{\max}} \geq c_{\mathsf{hard}}. \tag{2.4}$$

Example 2. *For $c_{\mathsf{hard}} = 80$, the following parameters meet these constraints:* $\gamma = 0.98, \ell = 13\,080, d_{\min} = 6\,000, d_{\max} = 6\,600, d = 11\,560, w = 99, d_Q = 20$.

2.3 Conclusion

Although the design of TCHO is very simple and well fitted for an hardware implementation, some major disadvantages are its prohibitive decryption time complexity, of exponential cost, and the absence of an estimate of the error probability in decryption. Experimentally, for some parameters suiting the assumptions, this probability is small, as predicted, but not enough to be neglected. Exhibiting an exact formula or even an approximation of the theoretical failure probability is difficult. A lower bound could be given if the minimum distance of a truncated LFSR code was known, but finding this distance is hard, and the bound one could obtain would anyway be too loose to be significative.

Chapter 3

Implementation of TCHO

In this chapter we review algorithms for an implementation of TCHO, and present the performances obtained. TCHO was implemented in C++, and compiled with `g++ 3.3.5`. We gained a precious time using Shoup's library for number theory [Sho05] (NTL), which efficiently implements all common operations in $\mathbb{F}_2[X]$, using a confortable and flexible representation. It was also used for computing polynomial factorization and `gcd`. Some help was also found in [Arn05, PTVF92]. All performances were measured on 1.5 GHz Pentium 4 computer.

3.1 Linear feedback shift register

When implementing LFSR's, the naive bit-per-bit approach is clearly uneefficient and very slow, especially for huge registers like ours, so we had to find a better algorithm.

3.1.1 Algorithm

Inspired from [CM03, CM01], this algorithm produces blocks of arbitrary size from a LFSR of any larger length. We first introduce some notations:

- b: the length of a block (in bits),
- n: the length of the LFSR,
- $m = \lceil \frac{n}{b} \rceil$: the number of blocks (*i.e* bitstrings of length b) used by the LFSR,
- P: the feedback polynomial, p_i its i-th coefficient, from zero (constant term) to n, and $P[i]$ denotes the i-th block, of b bits, of coefficients,
- $\mathcal{B}$: the new block we want to compute,

- S: the *state* of the LFSR; a sequence of blocks, $S[1], \ldots, S[m]$ of size b.
- $\boxplus$: bit-to-bit XOR operator,
- $\boxtimes$: bit-to-bit AND operator.

The algorithm is based on the leap-forward technique, whose basic idea is to build the block $\mathcal{B}$ by considering independently each tap, and recording the future bits located at its offset. The main procedure is composed of two stages:

1. build the block considering the taps involving only bits of the current state (*i.e.* taps over b),
2. finalize the block bit by bit while considering the taps $t_i \in [1, b]$ (we may need some of the first bits of $\mathcal{B}$ to build its last ones).

These steps respectively corresponds to the first two loops of the Algorithm 3.1, and the final loop aims just at updating the LFSR state. The taps in $[1, b]$ are treated separately since the future bits at their offset are not all known yet, and need to be computed dynamically.

Here the particular case of a LFSR of length non-multiple of b is implicitely handled, and the case with taps in the first block of the state leads to the second step described above.

3.1.2 Analysis

The time complexity of the Algorithm 3.1 to build a block of b bits is in $\mathcal{O}(\mathrm{w}_p(P))$; we loop over all the taps of P above b, and then make b iterations to process the first taps. In contrast, the naive bit-per-bit looks the d bits of the register to compute each output bit, so the algorithm runs in time $\mathcal{O}(bn) = \mathcal{O}(b \cdot \deg(P))$ when computing b bits, thus our algorithm requires in average $2b$ times less operations.

We did not found any better technique for software implementations of LFSR in the literature. Moreover, we used several precomputations, not mentionned in the Algorithm 3.1, to speed up the generation; we build two lists of the taps multiples and non-multiples of b, then treat them separately in the algorithm, it leads to a gain of about $3 \times d_P/32$ logical operations per block computed. The average number of elementary operations $(\ll, \gg, \boxplus, \boxtimes)$ required to build a block in our implementation is estimated to

$$n(2 - \frac{1}{2b}) + 6b.$$

In the particular case where the taps are all on a boundary (*i.e.* $p_i \neq 0 \Rightarrow b|p_i$), and when no one (except the constant term) has an offset lower than

Algorithm 3.1:

Input: S, P
Output: $\mathcal{B}$

1: $\mathcal{B} \leftarrow 0$
2: **for** $i = b+1, \ldots, n$ **do**
3: **if** p_i **then**
4: $\mathcal{B} \leftarrow \mathcal{B} \boxplus \left(S[\frac{i}{b}] \ll (-i \mod b)\right) \boxplus \left(S[\frac{i}{b}-1] \gg (i \mod b)\right)$
5: **end if**
6: **end for**
7: $k \leftarrow 1 \ll (b-1)$
8: **for** $i = 0, \ldots, b-1$ **do**
9: $buf \leftarrow (B[0] \ll i) \boxplus (\mathcal{B} \gg (b-i))$
10: **if** $\mathrm{w}_h(P[0] \boxtimes buf)$ is odd **then**
11: $\mathcal{B} \leftarrow \mathcal{B} \boxplus k$
12: **end if**
13: $k \leftarrow k \gg 1$
14: **end for**
15: **for** $i = m, \ldots, 1$ **do**
16: $S[i] \leftarrow S[i-1]$
17: **end for**
18: $S[0] \leftarrow \mathcal{B}$
19: **return** $\mathcal{B}$

b, this time complexity is reduced to

$$n(\frac{1}{2} + \frac{1}{b}).$$

Table 3.1 gives some examples of time complexities achieved with this algorithm (the field f.b. is ticked when there are no taps in the first block). We see that we had better using larger blocks for our large LFSR $\mathcal{L}_P$, and choosing small blocks combined with a trinomial without taps below the block size for the small LFSR $\mathcal{L}_Q$. However technical concerns must be considered: operations on the natural type of C++, `int` (32 bits) are much faster than on longer emulated types, and so we shall choose $b = 32$.

A C implementation of this LFSR algorithm independent of TCHO can be found at `http://www.131002.net`.

3.2 Pseudo-random generation with given bias

This is a big issue; weak pseudo-random generators (PRG) have often lead to unsecure systems in practice, even if it was secure on the paper, where the PRG is assumed to be ideal. We present several algorithms producing a

$\deg(P)$	$\mathrm{w}_p(P)$	b	f.b.	cost
20	3	8	$\surd$	6
20	3	8		61
6 000	3 000	32	$\surd$	3 187
6 000	3 000	32		12 098
6 000	3 000	64	$\surd$	3 093
6 000	3 000	64		12 337

Table 3.1: Number of operations to build a block.

bitstream of pseudo-random bits with a given bias from a source of uniform bits. The essential concerns are the time complexity, the number of uniform pseudo-random bits required to produce one bit, and the soundness with the theoretical bias, expressed as the statistical distance to the ideal distribution, with the assumption that the uniform generator used is ideal.

3.2.1 Choice of a random source

The candidates

The three following sources are suggested:

1. The `rand()` function of the C language, based on a linear congruential generator. Its seed is 32 bits long only, so there are only 2^{32} distinct bitstreams (note $32 < c_{\mathsf{easy}}$).

2. The PRG ISAAC [Jen96a]: *"ISAAC requires an amortized 18.75 instructions to produce a 32-bit value. There are no cycles in ISAAC shorter than* 2^{40} *values. The expected cycle length is* $2^{8\,295}$ *values"* [Jen96a]. It is designed to be cryptographically secure[1], but is *not proved* to be. The only attack published is a known plaintext one [Pud01], and runs in time complexity $4.67 \cdot 10^{1\,240}$. Its seed is 8 192 bits long.

3. The file `/dev/urandom`, physical entropy source on Unix systems.

The third proposal can already be dismissed: it cannot be seeded, thus we cannot reproduce a random string, and it requires I/O system calls, slowing the generation. Also, some weaknesses of this generator were pointed out by the analysis performed in [GPR06].

Statistical tests

We use the program ENT [Wal98], displaying minimal statistical results: it is a good tool to compare generators, but the criteria considered are quite

[1] A PRG can be declared *cryptographically secure* when it passes the next-bit test, *i.e.*, when no polynomial-time adversary can predict the k-th bit from the $k-1$ previous bits with probability greater than 1/2.

superficial, and cannot be used to valid cryptographic generators. We use samples of one megabyte (2^{23} bits). Table 3.2 presents the most significative results of ENT, and time records: we display the entropy of both bytes and bits, the error percentage in Monte-Carlo π estimation, the correlation coefficient (0 when totally uncorrelated), and the average time for computing one megabyte of pseudo-random bits.

PRG	ent.(bytes)	ent. (bits)	π error	correlation	time
`rand()`	7.999	1.0	0.30 %	$-5.7 \cdot 10^{-4}$	17 ms
ISAAC	7.999	1.0	0.04 %	$-2.8 \cdot 10^{-4}$	6 ms

Table 3.2: ENT results.

The Diehard battery of tests [Mar95a] is a set of empirical tests that must be passed by a cryptographically secure PRG, (here a huge period does not suffice, *cf.* the Mersenne Twister [MN98]): "Most of them seem to present a major leap in sensitivity to detect particular statistical defects in sequences of bits over the so called standard tests such as Chi Square, bias, various correlation tests, enthropy test, picturing randomness and so on. Diehard tests are thereefore often reffered to as stringent tests." [Mar95a]. Here samples of at least 2^{26} bits are required. ISAAC successfully passed all the tests, while `rand()` did not even passed one. Moreover, it is known that the lower-bits of numbers produced by a linear congruential generator are "less random" than higher-bits, and that linear congruential generators are far from being secure [Ste87].

Final choice

ISAAC is clearly a better PRG than `rand()`, moreover it is a cryptographically secure generator suitable for real applications, also used as a stream cipher in some cases. Both the algorithm and the implementation provided by its author are in public domain. We shall seed the generator using the file `/dev/urandom`.

Last minute addition: we found dramatic flaws in ISAAC, see Appendix A for details. To replace ISAAC we suggest to use the keystream generator QUAD [BGP06], both proved secure and practical, but requiring 1Mo of memory.

3.2.2 Algorithm G1

This is the algorithm suggested informally in [FV06]. The parameters are n, the length of the block produced, and B, the maximal precision allowed (*e.g.* it would be 32 using the type `float`, representing floating point numbers with precision less than 2^{-32}). The basic idea is to build a binary tree where the

leaves are some words of fixed length n. We first describe the precomputation steps, then the generation algorithm.

Precomputation

Given a bias $0 \leq \gamma \leq 1$, we have to build a rooted binary tree representing the probability law induced, to do this, we follow this procedure:

1. Compute the probability associated with each word. For a word of weight k, it is $p_\gamma^{n-k}(1-p_\gamma)^k$. This step is achieved in $\mathcal{O}(n)$ operations (the probability is computed once for all words of a given weight[2]).

2. Decompose each probability as a sum of inverse powers of 2, with a precision bound B (*i.e.* we do at most B divisions by 2, for a precision of 2^{-B}). It requires $\mathcal{O}(Bn)$ atomic operations.

3. Build the tree, where each leave is a word (not necessarily distincts), and a word appears at depth k if and only if its decomposition in powers of 2 contains 2^{-k}. This process is clearly deterministic, and runs in time $\mathcal{O}\left(2^B\right)$ (the maximal number of nodes of the tree).

It gives a global cost in $\mathcal{O}\left(Bn + 2^B\right)$. An example of tree is represented in Figure 3.1. There are at least 2^n leaves (as much as distinct words), and at most $B \cdot 2^n$.

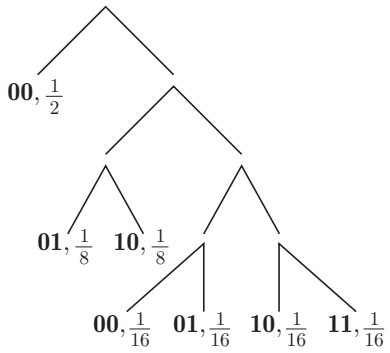

Figure 3.1: Tree of **G1** for $\gamma = \frac{1}{2}$, $B = 4$, $n = 2$.

[2]This requires the computation of $n/2$ binomial coefficients (since $\binom{n}{k} = \binom{n}{n-k}$). For an exact result, computing $n!$ requires $n-1$ integer products, and the computation of all the $k!$ does not requires additional cost, since recorded during the computation of n. Finally $n/2$ divisions are realized, and so the cost of computing all the binomials is in $\mathcal{O}(n)$ operations. However the number of bits in memory is in $\mathcal{O}(\log(n!))$.

Word generation

To pick a word ω, one just goes through the binary tree by successive coin flips until a leaf is met, as described in Algorithm 3.2 (r_i is the i-th bit of r, the function root return the root node of a tree, leftChild and rightChild return respectively the left or right child of a node, which is a node too, and the function label returns the word corresponding to a leaf). Since the number of nodes is finite, the algorithm always finishes. Its correctness follows from the decomposition of the probability distribution.

Algorithm 3.2:

Input: $r \in \{0,1\}^B$, T (the tree)
Output: $\omega \in \{0,1\}^n$

1: $currentNode \leftarrow \mathsf{root}(T)$
2: $offset \leftarrow 0$
3: **while** $currentNode$ is not a leaf **do**
4: **if** $r_i = 0$ **then**
5: $currentNode \leftarrow \mathsf{leftChild}(currentNode)$
6: **else**
7: $currentNode \leftarrow \mathsf{rightChild}(currentNode)$
8: **end if**
9: $i \leftarrow i + 1$
10: **end while**
11: $\omega \leftarrow \mathsf{label}(currentNode)$
12: **return** w

Complexity

The Algorithm 3.2 clearly runs in time $\mathcal{O}(B)$. In [FV06] the average number of unbiased bits required to build one biased word is lower than $n+2$, so the cost for a single bit is

$$R_{\mathbf{G1}} = 1 + \frac{2}{n}.$$

Example 3. *For* $n = 32$, $R_{\boldsymbol{G1}} = 1.0625$.

Statistical distance

Let W be the random variable of the word outputed by **G1**, $\Pr_{\mathbf{G1}}[X = \omega]$ be the probability that **G1** outputs the word ω, and $\Pr_I[W = \omega]$ the theoretical probability associated with that word. The statistical distance of **G1** from an ideal generator I is thus

$$D_{\mathbf{G1}} = \sum_{\omega \in \{0,1\}^n} \left| \Pr_{\mathbf{G1}}[W = \omega] - \Pr_I[W = \omega] \right|.$$

In average, the theoretical probability differs from the effective of $\frac{2^{-B}}{2}$, and so

$$D_{\mathbf{G1}} \approx 2^{n-B-1}.$$

3.2.3 Algorithm G1+

We remark that in the previous algorithm, words of equal weight have the same probability of occurence, thus we can propose an alternative algorithm, where the leaves are not words anymore, but weights in $\{0, \ldots, n\}$.

Precomputation

Although the asymptotic time complexity remains the same than for **G1**, the average one is reduce for the third step. Thus to build the tree we follow these steps:

1. Compute the probability associated with each weight. For the weight $k \in [0, n]$ this probability is (this follows a $(n, 1 - p_\gamma)$ binomial distribution)

$$p_\gamma^{n-k}(1 - p_\gamma)^k \binom{n}{k}.$$

   It requires $\mathcal{O}(n)$ operations, using a non-naive algorithm for binomials.

2. Decompose each probability, like for **G1**, still in $\mathcal{O}(Bn)$ operations.

3. Build the tree, in $\mathcal{O}\left(2^B\right)$ operations.

Now the tree has at least n leaves, and at most Bn, instead of 2^n and $B \cdot 2^n$.

Word generation

Like for **G1**, we first randomly pick a leaf of the tree, using Algorithm 3.2, where the word returned is now on $\lceil \log n \rceil$ bits instead of n (at most $\lceil \log n \rceil$ bits are required to code an integer in $[0, n]$). Then we randomly pick a word of the weight k found: we can use the Algorithm 3.3 (random(n) returns a random integer in $[0, n[$), or use another tree to pick the word.

Complexity

The word generation algorithm still runs in time $\mathcal{O}(n)$. Compared to **G1**, the number of leaves is exponentially reduced, thus the average cost to select a weight is about $\log n + 2$. To build a word of given weight, picking each offset one by one has an average cost of $n \log \frac{n}{2}$, but we reach a lower cost using

Algorithm 3.3:

INPUT: k (a weight)
OUTPUT: $\omega \in \{0,1\}^n$

1: $i \leftarrow 0, w \leftarrow (0,\ldots,0)$
2: **while** $i < k$ **do**
3: $offset \leftarrow \mathsf{random(n)}$
4: **if** $\omega_i = 0$ **then**
5: $\omega_i \leftarrow 1$
6: $i \leftarrow i+1$
7: **end if**
8: **end while**
9: **return** ω

a binary tree: there are in average $\binom{n}{n/4}$ possible words, thus the number of bits to choose one is about $\log \binom{n}{n/4} + 2$. It gives a total cost of less than

$$R_{\mathbf{G1}+} = \frac{\log\left(n\binom{n}{n/4}\right) + 4}{n}$$

for a single bit.

Example 4. *For words of $n = 32$ bits, $R_{\mathbf{G1}} = 1.06$, and $R_{\mathbf{G1}+} \approx 1.01$.*

Statistical distance

Let W be the word outputed by **G1**+. First observe the following fact: for a random word ω,

$$\Pr_{\mathbf{G1}+}[W = \omega | \mathrm{w}_h(X) = \mathrm{w}_h(\omega)] = \Pr_{I}[W = \omega | \mathrm{w}_h(X) = \mathrm{w}_h(\omega)].$$

That is, once a weight is picked, we assume that the algorithm randomly picking a word of this weight behaves as an ideal one. So the statistical distance from **G1**+ to an ideal generator is

$$
\begin{aligned}
D_{\mathbf{G1}+} &= \sum_{\omega\in\{0,1\}^n} \left| \Pr_{\mathbf{G1}+}[W=\omega] - \Pr_{I}[W=\omega] \right| \\
&= \sum_{\omega\in\{0,1\}^n} \Big(\Pr_{I}[W=\omega | \mathrm{w}_h(W)=\mathrm{w}_h(\omega)] \\
&\quad \times \left| \Pr_{\mathbf{G1}+}[\mathrm{w}_h(W)=\mathrm{w}_h(\omega)] - \Pr_{I}[\mathrm{w}_h(W)=\mathrm{w}_h(\omega)] \right| \Big) \\
&\approx 2^{-B-1} \sum_{\omega\in\{0,1\}^n} \Pr_{I}[W=\omega | \mathrm{w}_h(W)=\mathrm{w}_h(\omega)] \\
&= 2^{-B-1} \sum_{\omega\in\{0,1\}^n} \binom{n}{\mathrm{w}_h(\omega)}^{-1} \\
&= (n+1)\cdot 2^{-B-1}.
\end{aligned}
$$

Conclusion

Our variant of **G1** achieves a much better statistical distance, and reduced requirement in time an memory, however its construction is a bit more complex, and this may not be worth its cost for some hardware implementations. Like previously, this kind of algorithm operating on bits is tedious to handle in software, we will present another kind of algorithm reproducing the two stages of **G1**+ a bit differently.

3.2.4 Algorithm G2

Description

This is another block-oriented algorithm, based on the same idea than **G1**+. It produces a random block in two steps:

1. pick a random weigth k (following a $(n, 1-p_\gamma)$ binomial law),
2. pick a random word of weight k (uniformly).

The Algorithm 3.4 outputs a word ω of size n with respect to a bias $0 \leq \gamma \leq 1$; the function $\mathsf{distribution}(\gamma)$ returns a list of rational numbers $(o_i)_{i=-1,0,\ldots,n}$, $0 = o_{-1} < o_0 < \cdots < o_n = 1$, describing the weights probability distribution:

$$
o_i - o_{i-1} = p_\gamma^{n-i}(1-p_\gamma)^i \binom{n}{i}, \forall i \in 0,\ldots,n.
$$

The function $\mathsf{frandom}()$ returns a uniform rationnal number in $[0,1]$, with precision 2^{-B}. The probability distribution is precomputed, then the number of random bits required to produce a word of size n is roughly

- B to pick a weight k with precision 2^{-B}, $B > 1$,
- $k \log n$, to choose a word of weight k ($\log n$ bits are required to pick an offset in the word of length n).

Algorithm 3.4:

INPUT: n (block size), γ (a bias)
OUTPUT: $\omega \in 0, 1^n$

1: $(o_i)_{i=0,\ldots,n} \leftarrow$ distribution(γ)
2: $r \leftarrow$ frandom()
3: $i \leftarrow 0, weight \leftarrow -1$
4: **while** $weight < 0$ **do**
5: **if** $r < o_i$ **then**
6: $weight \leftarrow i$
7: **end if**
8: $i \leftarrow i + 1$
9: **end while**
10: $i \leftarrow 0, \omega \leftarrow (0, \ldots, 0)$
11: **while** $i < weight$ **do**
12: $offset \leftarrow$ random(n)
13: **if** $w_i = 0$ **then**
14: $\omega_i \leftarrow 1$
15: $i \leftarrow i + 1$
16: **end if**
17: **end while**
18: **return** ω

Complexity

The precomputation consists in partitioning the interval $[0, 1] \subset \mathbb{Q}$ into $n+1$ subintervals of magnitude

$$p^k (1-p)^{n-k} \binom{n}{k},$$

for $k = 0, \ldots, n$. This requires $\mathcal{O}(n)$ operations. The Algorithm 3.4 runs in time $\mathcal{O}(n)$. To produce a block of n bits with precision B and bias γ, the cost in terms of uniform random bits is

$$B + (1 - p_\gamma) n \lceil \log n \rceil$$

That is, for one bit, a cost of

$$R_{\mathbf{G2}} = \frac{B}{n} + (1 - p_\gamma) \lceil \log n \rceil.$$

The additional cost due to offsets collisions was neglected here, since negligible for our high biases. Indeed, the expected number of collisions when picking a word of weight $\bar{w} = (1 - p_\gamma)n$ is

$$\sum_{i=1}^{\bar{w}-1} \sum_{j=1}^{\infty} \frac{i \cdot j}{n^j} = \sum_{i=1}^{\bar{w}-1} \frac{i}{n(1-\frac{1}{n})^2} = \frac{(\bar{w}-1)(\bar{w}-2)}{2n(1-\frac{1}{n})^2}.$$

Example 5. *For $B = n = 32$ bits and $\gamma = 0.98$, $R_{\mathbf{G2}} = 1.05 \approx R_{\mathbf{G1}}$. The expected number of collisions is* 0.033*.*

Statistical distance

Like in **G1**+, we get

$$D_{\mathbf{G2}} \approx (n+1) \cdot 2^{-B-1}.$$

3.2.5 Algorithm G3

Description

We propose a different kind of generator, which does not produce blocks but directly a stream of fixed length, with a weight depending of the bias; for a sequence of ℓ bits, we will have in average $p_\gamma \cdot \ell$ zeros, and thus will build a sequence of weight exactly $\lfloor (1 - p_\gamma) \times \ell \rceil$. We will develop this idea in the following (this technique comes from an idea of Serge Vaudenay). The technique suggested above has another advantage: it would guarantee that the decoding will not get harder, *i.e.* that the sequence won't contain more ones than predicted in average. An obvious drawback is that it reduces the number of possible sequences (of length ℓ and bias γ) to

$$\binom{\ell}{(1-p_\gamma) \cdot \ell}.$$

We now describe a generic algorithm (Algorithm 3.5) producing a bitstring of length ℓ and weight p (here $\mathsf{random(n)}$ returns a uniform random integer in $[0, n[$, and ω_i still denotes the i-th bit of the word ω).

Complexity

The time complexity of Algorithm 3.5 is clearly in $\mathcal{O}(n)$, and we do n calls to the function `random`. At each loop, the value $p + q$ decreases from one, thus the total number of uniform random bits required is

$$\sum_{k=1}^{\ell} \lceil \log k \rceil \leq \ell + \lceil \log(\ell!) \rceil.$$

Algorithm 3.5:

INPUT: ℓ (stream length), γ (a bias)
OUTPUT: a word $v \in \{0,1\}^\ell$

1: $p \leftarrow \lfloor \ell \cdot p_\gamma \rceil$
2: $q \leftarrow \ell - p$
3: **for** $i = 1, \ldots, n$ **do**
4: $\quad j = \mathsf{random}(p + q)$
5: $\quad$ **if** $j < p$ **then**
6: $\quad\quad \omega_i \leftarrow 0$
7: $\quad\quad p \leftarrow p - 1$
8: $\quad$ **else**
9: $\quad\quad \omega_i \leftarrow 1$
10: $\quad\quad q \leftarrow q - 1$
11: $\quad$ **end if**
12: **end for**
13: **return** ω

That is, for one bit, a cost of less than

$$1 + \frac{\log(\ell!)}{\ell}$$

uniform pseudo-random bits.

Since TCHO shall require bitstring of small weight (high γ), we had better using the strategy, by picking offsets in the word, thus requiring

$$R_{\mathbf{G3}} = (1 - p_\gamma)\lceil \log \ell \rceil$$

uniforms bits per biased bit.

Example 6. *For* $\ell = 10\,000, \gamma = 0.98$, $R_{\boldsymbol{G3}} \approx 0.13 < R_{\boldsymbol{G2}}$.

Statistical distance

The difference between the theoretical probability and the probability induced by our construction to output a 0, is

$$D_{\mathbf{G3}} \approx \frac{\lfloor \ell \times p_\gamma \rceil}{\ell} - p_\gamma.$$

Flaw

The problem of distinguishing between a bitstream produced by **G3** and an ideal biased generator is trivial: one only has to count the ones in the stream, and pick the bit sequence which has exactly $\lfloor \ell \cdot p_\gamma \rceil$ zeros. The advantage is equal to the probability that the ideal random source does not match this exact value.

Example 7. *For $\gamma = 0.95$, $\ell = 100$, we should have $\lfloor \ell \times (1 - p_\gamma) \rceil = 3$ ones in the sequence produced by **G3**. An ideal biased generator deviates from this number with probability*

$$1 - p_\gamma^{97}(1 - p_\gamma)^3 \binom{100}{3} \approx 0.78,$$

which is the advantage of an adversary on the distinguishing problem.

Such a weakness is alarming for a PRG, and so we cannot use **G3** *a priori* in TCHO. But it could be part of a variant of the cryptosystem, taking advantage of the properties of this generator; for instance, it guarantees that the bias does not deviate too much.

3.2.6 Algorithm G4

Description

This generator mixes **G2** and **G3**: a bitstring of length ℓ is directly generated, by first choosing a weight, then a word of the chosen weight. It is equivalent to **G2** with a block size $n = \ell$.

Complexity

Like for **G2**, the precomputation requires the computation of $\ell/2$ binomial coefficients, achieved in $\mathcal{O}\left(\ell(\log \ell)^2\right)$ operations.

The number of random bits required for one biased bit is in average

$$R_{\mathbf{G4}} \approx \frac{B}{\ell} + (1 - p_\gamma)\lceil \log \ell \rceil,$$

by neglecting additional cost induced by the collision (*e.g.* for $\ell = 10\,000$ and $\gamma = 0.985$ we get in average of ≈ 0.0004 random bits per biased bit produced). Here the expected number of collision is (*cf.* Section 3.2.4) is less than

$$\frac{(1 - p_\gamma)^2 \cdot \ell}{2}.$$

The time complexity of the algorithm ouputing a biased bitstring of length ℓ is clearly in $\mathcal{O}\left(\ell(1 - p_\gamma)\right)$.

Statistical distance

Like **G2** the statistical distance to an ideal generator is

$$D_{\mathbf{G4}} \approx (\ell + 1) \cdot 2^{-B-1}.$$

Example 8. *For $\ell = 40\,000, \gamma = 0.985, B = 64$, the expected number of collisions is about 1.13, leading to an additional cost of about 0.0004 random bits per bit produced. Neglecting this cost, we get $R_{\boldsymbol{G4}} \approx 0.117$, and $D_{\boldsymbol{G4}} \approx 2^{-35}$. The theoretical number of random bits per biased bit required is (information entropy)*

$$-p_\gamma \log p_\gamma - (1-p_\gamma)\log(1-p_\gamma) \approx 0.081.$$

Optimality

Let $p = p_\gamma$. The function $\frac{B}{\ell} + (1-p)\log \ell$ has a unique minimum value, reached when

$$-\frac{B}{\log e} + (1-p)\ell = 0,$$

that is, for

$$\ell = \frac{B}{(1-p)\log e}.$$

For example, for the previous example we get $\ell = 5914$, and a cost $R_{\mathbf{G4}} \approx 0.108$.

By substituting in the expression of $R_{\mathbf{G4}}$, we get

$$(1-p)(\log B - \log(1-p)),$$

which reaches the theoretical optimal value (the theoretical amount of information contained in a bitstring, that is, the binary entropy of p, *cf.* Section 1.2) when

$$\log B = \frac{p \log p}{p-1}.$$

Since the maximum of $\frac{p\log p}{p-1}$ on $]0,1[\subset \mathbb{R}$ is

$$\lim_{p\to 1} \frac{p\log p}{p-1} \approx 1.442,$$

a quasi-optimal cost will not be achieved for large enough values of B. In particular, if we tolerate a statistical distance as large as $(\ell+1)\cdot 2^{-B-1} = 1$, we can choose $B = \log \ell$, and so the cost reaches the theoretical minimal value when

$$\log\log \ell - \frac{p\log p}{p-1}$$

is close to zero, which requires $\ell = 3$ to be below zero.

In the case where a large number of biased bits is required, one often had better choose $\ell = B/((1-p)\log e)$ as "block size", to get the lowest cost in terms of random bits.

To conclude, this generator is very cheap in terms of random bits required, but can never reach a quasi-optimal cost. One may be careful to

the statistical distance induced (*i.e.* $B = 32$ would lead to a high distance for common TCHO2 parameters). Compared to **G2**, the precomputation is much more costly: computing $\ell!$ naively is required for an exact estimation of the probability distribution, and division of numbers on about ℓ bits must be performed; we had better use the following Algorithm 3.6, used in the GMP library [Gra06], based on the recurrence relation

$$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}.$$

It requires less than ℓ^2 products and divisions of numbers on $\log \ell$ bits.

Algorithm 3.6:

Input: $n, k, k \leq n/2$
Output: $\binom{n}{k}$
1: $b \leftarrow n - k + 1$
2: **for** $i = 2, \ldots, k$ **do**
3: $\quad b = b \times (n - k + i)$
4: $\quad b = b/i$
5: **end for**
6: **return** b

If we tolerate a slight loss of precision, we can first estimate $\ln(n!)$, $\ln(k!)$, and $\ln((n-k)!)$ using the Gamma function, defined by the integral:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt.$$

Hence for a natural n, $\Gamma(n) = (n-1)!$. The value $\ln \Gamma(n+1)$ can be approximated in constant time [PTVF92], thus approximating a binomial coefficient requires a constant number of multiplications and exponentiations (the exponentiation of e can be reduced to an exponentiation of the integer 2 to an integer power, with three additional products of small rational numbers, so as to avoid expensive floating point arithmetic). For instance we computed 2 000 000! in about three seconds, whereas the best exact algorithms using a database of prime factors (Schoenhage, Luschny) takes about one minute for 2 000 000!, and the best without a prime factors (split recursive) list takes about two minutes (see [Lus06]).

3.2.7 Conclusion

We overviewed several algorithms for biased random generation: **G1** has a somewhat low cost in terms of unbiased bits, but is not well suited for a software implementation, like its variant **G1**+. The algorithm **G2** is more software-friendly, for roughly the same cost in time and random bits. The

Algorithm **G3** is nsecure, but requires only a few uniform random bits, and its properties may be exploited by some variant of TCHO2. The last algorithm **G4** requires less uniform random bits than bits produced, with a low-complexity and simple implementation. A precision bound $B = 64$ or larger shall be required, depending on the values of ℓ. However the precomputation is much more expensive than for **G2**, so we will finally use the latter, since we can reasonnably allow us a higher number of random bits from ISAAC, and we may encounter large values of ℓ, where **G4** exact precomputations may become too costly.

3.3 Primitivity testing of a high-degree polynomial

Here we briefly study the problem of testing the primitivity of a high-degree polynomial, since required in the key generation procedure.

Testing primitivity of a binary polynomial is hard. We recall that an irreducible binary polynomial of degree d is primitive if and only if its order is equal to $2^d - 1$. Otherwise, its order is a divisor of $2^d - 1$ (and so is odd). Thus testing primitivity by computing the order is as hard as finding a factor of $2^d - 1$. Indeed, the three problems of computing order of an element in a group, finding and recognizing a primitive element in a finite field, are all listed as open problems in [AM94]. But in our particular case of polynomials over $\mathbb{F}_2$, maximal orders are Mersenne numbers, which have some properties relative to primality testing and factoring; we present some results about these numbers, deduce selection criteria for our polynomial P, factor of K. We start by an estimation of the primitive polynomials proportion.

3.3.1 Proportion of primitive polynomials

The proportion of primitive polynomials among the irreducibles is approximated in [FV06] to $8/\pi^2 \approx 81$ %, where π is introduced via Buffon's needle problem. So the probability for K to have a primitive factor of degree d_P is about $\frac{8}{\pi^2 d_P}$. Given an interval $[d_{\min}, d_{\max}]$, we get a probability of

$$1 - \prod_{i=d_{\min}}^{d_{\max}} (1 - \frac{8}{\pi^2 \times i})$$

that K has a factor of degree in this interval.

However, we can doubt of the accuracy of the estimation $8/\pi^2$: the exact enumeration formula (see Appendix C) give a proportion of primitive polynomials among irreducible, for degrees below 100, of 70 %. Unfortunately we cannot compute this value for the large degree ranges, since the full factorization is required in the formula (to compute Euler's totient and Mőbius functions). Indeed, the probability that a random irreducible polynomial of

degree d is primitive is exactly

$$\frac{\phi(2^d-1)}{\sum_{m|d}\mu\left(\frac{d}{m}\right)2^m}.$$

The number of irreducible polynomials of degree d can be sharply approximated to $2^d/d$, but there is no way to simply estimate the number of primitive ones.

3.3.2 Known deterministic tests

When 2^d-1 is prime, we know for sure that any irreducible polynomial of degree d is primitive (Corollary 1). Thus by testing for M_d primality, we can trivially prove primitivity in some cases. However, prime Mersenne numbers are seldom, and it suffices to store in memory the exponents of primes (they are only two exponents of Mersenne primes in $[5\,000, 1\,000]$: 9 689 and 9 941), and we do not have to use primitivity tests. In the case where primality has to be tested, several properties of Mersenne numbers may help.

When 2^d-1 is composite, the naive method is:

1. Get the full factorization $2^d-1=\prod_{i=2}^{\infty}p_i^{\alpha_i}$, where p_i is the i-th prime, and only a finite number of α_i is non-zero.

2. Compute $X^{(2^d-1)/k} \mod P$, where k ranges over all the prime factors of 2^d-1 until we find 1 (otherwise P is primitive).

Rieke *et al.* [RSP98] improved this generic algorithm, but finding a single factor of 2^d-1 still has a superexponential time cost in d with the best known algorithms, thus it is clearly infeasible for our degree ranges ($d > 6\,000$). Matsumoto [MN98] builds another kind of algorithm for the binary polynomials, based on bit to bit operations, but again it is too costly to be applyed in TCHO. All the other known methods require factoring 2^d-1 or computing discrete logarithms in $\mathbb{F}_{2^d}$, both notoriously difficult.

There exist fast deterministic techniques to compute elements of high order in some finite fields [GvP98], but they are not suitable in our case. Some works focus on trinomials, and algorithms were built to find "almost primitive" high degree trinomials [BLZ03, BZ03] . So there is no magical test avoiding the order computation, even if, as we see further, Mersenne numbers induce a slight advantage.

When d is prime, the following theorem gives a criterion on the prime factors of M_d:

Theorem 5 (Fermat, Euler)**.** *Let p and q be odd primes. If p divides 2^q-1, then $p \equiv 1 \mod q$ and $p \equiv \pm 1 \mod 8$.*

Proof. If p divides $2^q - 1$, then $2^q \equiv 1 \mod p$, and the order of 2 in $(\mathbb{Z}_p)^\star$ divides q, thus it must be q, because it is prime. By Fermat's Little Theorem, the order of 2 in $(\mathbb{Z}_p)^\star$ divides $p-1$, so $p - 1 = 2qk$. It gives

$$2^{(p-1)/2} = 2^{kq} \equiv 1 \mod p$$

so 2 is a quadratic residue modulo p, and it follows $p \equiv \pm 1 \mod 8$, which completes the proof. □

This result can be used to adapt the Pollard's *p-1* algorithm, and Eratosthene's sieve. When d is not prime, the Elliptic Curve Method is well adapted, but not efficient enough to get the full factorization in a reasonnable time for our ranges of exponents.

3.3.3 Using a non-primitive polynomial

Let's consider the case where P is of unknown order: in [FV06] the primitivity quality is required so as not to have $X^n - 1$ as a trivial solution, when the order is $n \leq d$, and a period long enough. The period would be shorter than ℓ with probability about ℓ/M_d, which is close to zero.

If P is not primitive, and the order n known, it is less than d with probability about d/M_d, which is also close to zero. So the trivial solution cannot be used. But one may factorize its order. For instance, if $n = 3p$, we can build a multiple of weight 3 and degree $2n/3$, but the probability that this number is lower than ℓ is close to zero again. A result on the factorization of $X^n - 1$ may be used:

Theorem 6. *Considering polynomials in the ring $\mathbb{F}_p[X]$, with p prime,*

$$X^n - 1 = \prod_{m|n} \Phi_m(X)$$

where the m-th cyclotomic polynomial is defined by

$$\Phi_m(X) = \prod_{d|m} (X^d - 1)^{\mu(m/d)}$$

with μ is the Mőbius function (m is not necessarily prime).

Proof. The result follows from the Mőbius inversion formula (see [LP98] Ch. 3, §13). □

The order n has an expected value close to 2^{d-1} (under the reasonnable assumption that orders are roughly uniformly distributed in $[3, 2^d - 1]$). When possible, exploiting the previous result would need to get the full factorization of n, and compute a number in $\mathcal{O}\left(2^{\nu(n)}\right)$ of combinations of the factors to hope finding some "good" multiple, which is infeasible for our values of d ($\nu(n)$ is the number of divisors of n).

We conclude that a non-primitive polynomial can be use with no significative risk.

3.3.4 A filter for primitive polynomials

Here we briefly describe a filter for non-primitive polynomials. In the following P is a randomly chosen irreducible polynomial, non-primitive, of arbitrary degree d chosen among non-Mersenne prime exponents (*i.e.* $2^d - 1$ is composite). A polynomial passing this test would be declared *probably primitive*. Our test is based on the following trivial property:

Property 1. $\forall k \in \mathbb{Z}, X^{k \cdot ord(P)} \equiv 1 \mod P$.

So if $X^{\frac{2^d-1}{k}} \not\equiv 1 \mod P$ for a given prime k less than $2^d - 1$, we know that k divides the order of P. Conversely, if it is 1 modulo P, then P is not primitive, and $\nu_k(2^d - 1) - 1 \geq \nu_k(\text{ord}(P))$, where, $\nu_k(n)$ is the multiplicity of the prime k in n. If $2^d - 1$ is square free, then $X^{\frac{2^d-1}{k}} \equiv 1 \mod P$ implies $k|\text{ord}(P)$. It is conjectured [Guy94] that all prime exponent Mersenne numbers are square free, but we cannot use this result, since primes are seldom.

The idea of our algorithm is to look for small prime factors of M_d, up to a certain bound B, and check $X^{\frac{2^d-1}{k}} \not\equiv 1 \mod P$ with k ranging over all these factors, so that we find 1 only if P is not primitive.

Algorithm The algorithm T is simply this procedure:

1. Find out all the distinct prime factors $p_1, \ldots, p_r$ of $2^d - 1$ less than B.
2. For $i = 1, \ldots, r$:

   If $X^{\frac{2^d-1}{p_i}} \mod P = 1$, then return 0.
3. return 1.

Correctness and complexity This algorithm is clearly deterministic. Trivially, for a random P, if P is primitive, then $T(P) = 1$. If P is not primitive, $T(P) = 1$ if and only if P's order has all the p_i's as factors. Let ρ be the probability that $T(P) = 1$ for a non-primitive P.

All the p_i can be found in time $\mathcal{O}\left(\sqrt{B} \cdot d^2\right)$ using Pollard's rho method (exponential cost in terms of the input's length) . There are at most d such factors, and in average $\ln \ln B$. Computing all the $X^{\frac{2^d-1}{p_i}} \mod P$ thus requires $\mathcal{O}\left(d^3\right)$ operations in the worst case (and $\mathcal{O}\left(d^2 \ln \ln B\right)$ on average). What remains to find is a bound on ρ.

Reliability It is known [HR17] that the average number of prime factors of an integer n is in average $\ln\ln n$, and $\ln\ln B$ for factors less than B (for high B), however Mersenne numbers may not behave like arbitrary integers; in 1964 Gillies [Gil64] made a conjecture about the distribution of prime divisors of Mersenne numbers, based on this Wagstaff [Wag83] estimates the expected number of prime factors of M_d between A and B to

$$\sum_k \frac{1}{k\ln(2kd)} \approx \ln\ln B - \ln\ln A$$

where the sum extends over all integers k such that $A < 2kd+1 \leq B$. Note that for $A = e$, we find the average estimate for arbitrary integers. Thus the expected number of distinct prime factors less than B is

$$\ln\ln B - \ln\ln 3$$

since M_d is odd, and the expected number of prime factors of M_d is

$$\ln\frac{\ln M_d}{\ln 3} \approx \ln d - \ln\ln 2 - \ln\ln 3.$$

Estimating the failure probability from an assumption on the orders distribution is far from being trivial, so we will use a more algebraic approach of the problem.

In the decomposition field $\mathbb{F}_2[X]/\langle P\rangle$, has its d distincts roots $(\dot{X},\ldots,\dot{X}^{2^d-1})$. This field has 2^d elements, and thus is isomorphic to $\mathbb{K} = \mathbb{F}_{2^d}$, and P roots are $\theta,\theta^2,\ldots,\theta^{2^d-1}$ for a certain θ. The set $(1,\theta,\theta^2,\ldots,\theta^{d-1})$ is a linearly independent family, and spans $\mathbb{K}$ as a d-vectorial space.

By definition, P is primitive if and only if each of its roots generates $\mathbb{K}^\times$. In that case all its roots are also generators of $\mathbb{K}^\times$, since θ's order is maximal, and they form a set stable by the Frobenius automorphism.

Let $F \subset \mathbb{K}$ be the set of all the θ such that $(1,\theta,\theta^2,\ldots,\theta^{d-1})$ is a linearly independent family. Elements of $\mathbb{K}\setminus F$ have their minimal polynomial of degree $d' < d$, and span a subfield of $\mathbb{K}$; hence their order divides $2^{d'}-1$, and $d'|d$. Conversely, if the order of θ divides $2^{d'}-1$ for some $d'|d$, then $\theta \notin F$. We have

$$\#\{\theta\in\mathbb{K}^\star, \mathrm{ord}(\theta)|2^{d'}-1\} \leq \frac{2^d-1}{2^{\sqrt{d}}-1},$$

so the fraction of $\theta\in\mathbb{K}^\star$ not in F is less than $\log d/(2^{\sqrt{d}}-1)$.

The probability that a random θ, such that θ^{2^d-1/p_i} for $i=1,\ldots,r$, is not a generator of $\mathbb{K}$, is the probability that a random $\alpha\in\mathbb{Z}/2^d-1\mathbb{Z}$ is not invertible given the fact $\mathsf{gcd}(\alpha,p_i)=1$. This probability is less than $\frac{d}{B\log B}$, let A be this event. We deduce

d	B	r	ρ
7000	2^{20}	37	$\leq 3.3 \cdot 10^{-4}$
7000	2^{30}	≈ 48	$\leq 2.2 \cdot 10^{-7}$
7003	2^{20}	2	$\leq 3.3 \cdot 10^{-4}$
7003	2^{30}	≈ 4	$\leq 2.2 \cdot 10^{-7}$

Table 3.3: Filter failure probability.

$$\Pr[\mathsf{NG}(\theta)|A] \leq \frac{d \cdot p_1}{B \log B},$$

where NG is the predicate "not a generator".

Finally,

$$|\Pr[\mathsf{NG}(\theta)|A, \theta \in F] - \Pr[\mathsf{NG}(\theta)|A]| \leq \Pr[\theta \notin F] \leq \frac{\log d}{2^{\sqrt{d}} - 1},$$

and so

$$\rho = \Pr[P \text{ not primitive}|T(P) = 1] \leq \frac{d}{B \log B} + \frac{\log d}{2^{\sqrt{d}} - 1}.$$

For common values of d in TCHO2, A negligible failure probability can be reached (note that the bound is not tight, since we considered the case of d factors, whereas they are only $\ln \ln B$ in average), *cf.* Table 3.3.

This algorithm easily generalizes for polynomials over the field $\mathbb{F}_p$, with p prime.

3.3.5 Conclusion

We have shown that testing primitivity was as hard as factoring a Mersenne number, which is an infeasible task for our exponents, and so we cannot deterministically test for primitivity. Moreover, no probabilistic polynomial time technique is known at this day to test primitivity. We suggested a deterministic filter, that finds with high probability non-primitive polynomials when the bound is sufficiently large. We also saw that even with an oracle returning the order of P and the full factorization of M_d, the probability of exploiting these values is clearly negligible.

Finally, we choose not to test primitivity at all, but we still have to check that P has no common factor with Q, for the decoding not to be ambiguous.

3.4 Key generation

We follow the process described in Section 2.2.1: first pick a random K of given degree and weight, then look for an irreducible factor of degree in

$[d_{\min}, d_{\max}]$. When such a polynomial is found, we have to test whether it is primitive or not; the probabilistic filter mentionned in the previous section could be use, but we also show that an irreducible P coprime with Q suffices, thus testing primitivity is not necessary.

We first perform the square-free factorization of K, which is a straightforward operation for polynomials: we start by computing the gcd of K and its derivate, then recursively build a decomposition of the form

$$K = \prod K_i^i$$

where the K_i are pairwise coprime square-free polynomials. At this stage we look for a suitable P, but finding our factor here will seldom occur, regarding to the parameters used. Then we apply the Cantor-Zassenhaus algorithm [CZ81b], the distinct degree factorization, to get the full factorization of K into powers of irreducible factors, from the square-free factors. This is a probabilistic algorithm for factoring on finite fields, of asymptotic time complexity in $\mathcal{O}\left(n^{2+o(1)} \ln 2\right)$, which is the best asymptotic complexity for a factorization algorithm today (the best deterministic algorithm runs in [Sho90] $\mathcal{O}\left(q^{1/2}(\ln q)^2 n^{2+o(1)}\right)$, where q is the cardinality of the finite field).

Example 9. *It takes on average about* 2 *seconds to get the full factorization of a polynomial of low-weight degree* 5 000*, and between* 20 *and* 30 *seconds for a polynomial of degree* 11 560 *(default parameter of TCHO).*

The public and private keys, represented as bitstrings, are respectively of length $(d_P + 1)$ and $(d + 1)$ bits. To reduce this length, one can store the offsets of the non-zero coefficients, it requires $w\lceil \log d \rceil$ bits. If coding numbers on an arbitrary number of bits is not practical (*e.g.* in software), one can store the polynomial K/P on $d - d_P$ bits, then recover K with one polynomial multiplication (cost in $\mathcal{O}\left(d_P^2\right)$).

Example 10. *A polynomial K of degree* 13 000 *and weight* 99 *can be stored on* $99 \times \lceil \log 13\,000 \rceil = 1\,386$ *bits, instead of* 13 000 *naively.*

3.5 Encryption and decryption

A ciphertext is the XOR of three bitstreams; $\mathcal{S}_{\mathcal{L}_P}$, $\mathcal{S}_{\mathcal{L}_Q}$, and $\mathcal{S}_\gamma$. In our implementation, these streams are arrays of 32 bit words, which are first computed independently, then xored word by word (there is a total of $\lceil \ell/32 \rceil$ words).

Decryption is not as easy as the encryption: the bitstream $K \otimes y$, of length $\ell - d$, is computed using bit operators on low-level representations of the stream and K in time $\mathcal{O}\left(d \cdot (\ell - d)\right)$. The matrix $\mathcal{M}_{f^{-1}}$ is precomputed, and its inversion is performed with a function of the NTL, implementing the

Gauss-Jordan algorithm. The product by this matrix is performed after the MLD, with an algorithm running in time $\mathcal{O}\left(d_Q^3\right)$.

Although TCHO is clearly a stream cipher (see Subsection 1.3.1), we meet a problem inherent to the block ciphers, when the message's length is not a multiple of the block size. A broadly solution is to systematically add a one to the message, then add zeros until a block is filled. It has the drawback to add one block of data to the cipher of messages whose length is a multiple of the block size, and thus can induce an expansion of the message. The *ciphertext stealing* technique can solve this problem when the block size is the same for plain and cipher messages, it is not the case here. So we have to use the first solution.

3.6 Experimental results

This section gathers practical information about our implementation, and benchmarks' results, based on the final version of the program. Table 3.4 gives the average time required to compute one a bitstream from a LFSR of one megabyte, and $\ell = 15\,000$ bits, in seven different scenarios, depending on the feedback polynomial:

I degree 30,

II degree 6 000,

III degree 6 000, with only taps on block boundaries,

IV degree 6 000, sparse (weight 50).

scenario	1 Mb	ℓ bits	rate
I	290.0 ms	452 μs	3 530 Kb/s
II	6.8 s	11.0 ms	150 Kb/s
III	1.1 s	2.1 ms	930 Kb/s
IV	1.0 s	1.8 ms	1 024 Kb/s

Table 3.4: LFSR performances.

Table 3.5 gives average time required to compute ℓ bits for $\ell = 15\,000$ and $\ell = 50\,000$, along with the rate achieved, using algorithm **G4** for different biases.

An alternative approach to compute a LFSR output is to use a precomputed look-up table: given a polynomial P of degree d_P, we can compute a table of $\ell \cdot d_P$ bits, containing the bitstreams produced by each initial state of $\mathcal{L}_P$ of weight equal to one. Computing such a table takes less than a second using optimized algorithms, then the generation of a bitstream requires

roughly $\frac{\ell}{32} \times \frac{d_P}{2}$ XOR operations (in our implementation, with a 32 bits processor). Experimentally the time gain is not significant, since memory access takes a non negligible time (about 70 megabytes are precomputed for common parameters).

γ	ℓ	time	rate
0.98	50 000	93 μs	64 Mb/s
0.98	15 000	42 μs	42 Mb/s
0.60	50 000	1 400 μs	4 Mb/s
0.60	15 000	440 μs	4 Mb/s

Table 3.5: PRG performances (using **G4**).

In Table 3.6 we present three sets of parameters satisfying the security constraints, and show in Table 3.7 the time required for the full key generation, the number of trials (number of candidates for K tried), the time for a full factorization, and for encryption and decryption.

scenario	d_Q	d_P	γ	w	d	ℓ	c_{hard}
I	16	$\in [5\,600, 6\,200]$	0.98	87	11 800	12 600	80
II	20	$\in [6\,000, 6\,600]$	0.98	99	11 560	13 080	80
III	20	$\in [7\,000, 7\,700]$	0.98	105	13 950	15 900	80

Table 3.6: Scenarios.

scenario	key generation	trials	fact.	encryption	decryption
I	160 s	6	19 s	12 ms	3 s
II	270 s	12	23 s	12 ms	68 s
III	290 s	8	38 s	12 ms	87 s

Table 3.7: Key generation and encryption performances.

The high values for decryption are due to the exponential cost in d_Q of the MLD. The time necessary for the matrix inversion is neglectable regarding to the cost of the MLD, for these parameters. This implementation does not use the improvement of the Walsh transform, which should reduce the theoretical time complexity of a factor $\frac{d_Q}{\ell - d}$, but may require non negligible additional computations.

Chapter 4

The TCHO2 scheme

We present a variant of TCHO, resulting of our study: we first show what kind of codes can be used to encode the message, and suggest much better codes than arbitrary LFSR ones for our encryption scheme. Another innovation of TCHO2 is that the need for P to be primitive is obviated (*cf.* discussion in Chapter 3).

4.1 Presentation

TCHO2 differs from TCHO in the coding applied to the plaintext. In TCHO, a code spanned by an LFSR with an arbitrary primitive polynomial Q was used, leading to an expensive decryption procedure. In TCHO2 we will instead use a code C of dimension k and length ℓ for which an efficient decoding procedure exists, and denote $C(x)$ the codeword of x in C. This code is subject to many constraints and cannot be chosen at random. In the decryption process of TCHO, the ciphertext is multiplied by K to cancel $\mathcal{S}^{\ell}_{\mathcal{L}_P}$. In this process, the noise source $\mathcal{S}^{\ell}_{\gamma}$ becomes $\mathcal{S}^{\ell-d}_{\gamma^w}$, but $\mathcal{S}^{\ell}_{\mathcal{L}_Q(x)}$ also becomes $\mathcal{S}^{\ell-d}_{\mathcal{L}_Q(x')}$. In the case of TCHO2, the multiplication by K being a linear operation, we will have $K \otimes C(x) = \tilde{C}(x)$, where $\tilde{C}$ is a new linear code of dimension k and length $\ell - d$. This means that when decrypting a ciphertext, one will have to decode in the modified code $\tilde{C}$. The only case where decoding in $\tilde{C}$ can be efficient for any K is when C is a truncated cyclic linear code, that is, C is the output of an LFSR. In that case, as for TCHO, $K \otimes C(x)$ is equal to $C(x')$ truncated to $\ell - d$ bits, where x' is obtained from x exactly as with TCHO. TCHO2 is thus at the same time a generalization of TCHO as things are seen from a more general point of view, but also a particular case as the only efficient solutions were already included in the scope of the original TCHO.

TCHO2 encrypts a plaintext x in the following way:

$$\text{TCHO2}_{\mathsf{enc}}(x, r_1||r_2) = C(x) + \mathcal{S}^{\ell}_{\mathcal{L}_P(r_1)} + \mathcal{S}^{\ell}_{\gamma}(r_2).$$

Let y be a ciphertext of some plaintext x. Decryption works as follows:

1. K is used to delete $\mathcal{S}_{\mathcal{L}_P}$ in y:

$$K \otimes y \approx \tilde{C}(x) + \mathcal{S}_{\gamma^w}^{\ell-d} = y'$$

   where $\tilde{C}(x)$ is equal to a truncated codeword $C(x')$, with $x' = f(x)$ for a certain linear application f.

2. y' is decoded to find x', and $x = f^{-1}(x')$ is recovered.

Note that the matrix of f^{-1} can still be precomputed, since it only depends on K and the code C used.

4.2 LFSR codes with trinomials

A first proposal, by Willi Meier, was to use, instead of an arbitrary primitive polynomial, a trinomial as feedback polynomial of the LFSR encoding the plaintext. In that case, decoding algorithms more efficient than MLD exist; the Algorithm B in [JJ99] or Gallagher decoding as used, *e.g.*, in [Wag02] for fast correlation attacks, can be applied. The success probability of these algorithms depends on the correlation value p_{γ^w}, and the ratio between the length of known output and the size of the LFSR for which the initial state is searched for. Again, concerning the reliability of these iterative algorithms, only experimental results seem to be available. For trinomials it can be seen from Table 3 in [JJ99] that, for example, correct decoding is expected if the known output has length 100 times the LFSR-length, and p_{γ^w} is 0.6 or larger. This clearly improves the complexity of the decoding, but we see in the next section that it can be reduced again.

4.3 Block repetition codes

4.3.1 Description and reliability

These codes offer straightforward encoding and decoding algorithms: for a block repetition code of dimension k and length $\ell = mk$, the codeword of a bitstring x of length k is formed of m contiguous repetitions of x, and so the minimum distance of the code is m (m is also equal to the expansion coefficient). Decoding is performed using majority logic decoding (MJD), which is equivalent to MLD for these codes, but runs in time $\mathcal{O}(\ell - d)$, instead of $\mathcal{O}\left(k \cdot 2^k\right)$. This complexity gap allows to encrypt blocks larger than c_{easy}, and even any length less than $\ell - d$. Note that using a repetition code is equivalent to setting $Q = X^{d_Q} + 1$ in TCHO (with $d_Q = k$).

Here $\tilde{C}$ has minimum distance $m' = \lfloor (\ell - d)/k \rfloor$, but decoding more than $\lfloor (m' - 1)/2 \rfloor$ errors (the theoretical bound for deterministic error correction)

will be possible. The probability of erroneous decoding is exactly the probability that at least one bit is more frequently erroneous than correct, that is (if we assume that the correlation in $\mathcal{S}_{\gamma^w}$ induced by the deletion of $\mathcal{S}_{\mathcal{L}_P}$ has no consequence here):

$$\rho \approx 1 - \left(\sum_{i=\lceil m'/2 \rceil}^{m'} p_{\gamma^w}^i (1 - p_{\gamma^w})^{m'-i} \binom{m'}{i} \right)^k . \tag{4.1}$$

This probability can also be expressed using the central limit theorem (summing k times on the m' bits). If τ is the random variable of the number of errors on a single bit, the probability that an error occurs in the decoding of this bit is $\Pr[\tau > m'/2]$, which is equal to

$$\Pr\left[\frac{\tau - m'(1 - p_{\gamma^w})}{\sqrt{m' p_{\gamma^w}(1 - p_{\gamma^w})}} > \sqrt{m'} \frac{p_{\gamma^w} - \frac{1}{2}}{\sqrt{p_{\gamma^w}(1 - p_{\gamma^w})}} \right] \approx 1 - \varphi(\sqrt{m'}\eta)$$

with $\eta = \gamma^w / \sqrt{1 - \gamma^{2w}}$. And so the failure probability obtained is

$$\rho \approx k \cdot \varphi(-\eta\sqrt{m'}). \tag{4.2}$$

Here φ is the cumulative distribution function of a standard normal distribution:

$$\varphi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt.$$

If the generator **G3** was used, the error probability would be expressed differently; if $\zeta = \lfloor (\ell - d)(1 - p_{\gamma^w}) \rceil$ is the exact weight of the pseudo-random bitstring, then the probability that given bit is badly decoded becomes

$$\nabla = \binom{\ell}{\zeta}^{-1} \cdot \sum_{i=0}^{\lceil m'/2 \rceil} \binom{m}{i} \binom{\ell - m'}{\zeta - i},$$

hence the probability of bad decoding is

$$\rho' \geq 1 - (1 - \nabla)^k.$$

The value obtained is just a lower bound, since it considers the k bitstrings of length m' independently, whereas they are not. Even so, this bound is close to the exact value, and experimentally it is also close to the value of ρ found in (4.2).

We can now ask the question: what error probability can we accept ? We must be careful in that choice, indeed a value as low as 2^{-23} ($\approx 10^{-16}$) looks small enough, but it implies in average one error for 2^{23} blocks of length k, that is, for $k = 64$, an expectancy of one error for 64 megabytes of data encrypted. In 1943, the mathematician Emile Borel informally introduced

four different scales [Bor43], to state that a given probability is *negligible*; at the *terrestrial scale*, even 1/1000 is negligible, but at the *cosmic scale* we should only neglect a probability lower than 10^{-50}. He defines an event with negligible probability as one *"which shall never happen, or, at least, shall never be observed"*[1]. In our case, we must already make some assumptions: will TCHO2 be used daily to cipher dozens of hard disks, or only monthly to encrypt 128 bits of some secret key ? The second affirmation sounds more realistic, regarded to the cost in space and time of TCHO2. We should also remember that requiring an error probability smaller than the one of hardware failure would be somewhat stupid. So we should be able to tolerate a failure probability of 10^{-10} at our "cryptographic scale", that is, a wrong decryption of a block every 500 Mb of data encrypted, or one key of 128 bits over 100 000 000.

	k	d_P	d	w	γ	$1 - p_{\gamma^w}$	ℓ	ρ
I	32	$\in [6\,600, 7\,200]$	13 470	89	0.9832	0.39	32 000	$1.0 \cdot 10^{-6}$
II	64	$\in [9\,000, 9\,900]$	17 550	97	0.9877	0.35	30 000	$4.0 \cdot 10^{-4}$
III	128	$\in [5\,900, 8\,200]$	24 420	51	0.9813	0.31	48 000	$2.9 \cdot 10^{-6}$
IV	128	$\in [5\,600, 10\,400]$	20 300	83	0.9837	0.37	62 000	$1.7 \cdot 10^{-4}$
V	128	$\in [8\,500, 9\,075]$	17 996	81	0.9870	0.36	68 000	$7.0 \cdot 10^{-11}$
VI	128	$\in [5\,800, 7\,000]$	25 820	45	0.9810	0.29	50 000	$8.9 \cdot 10^{-9}$

Table 4.1: Examples of parameters for TCHO2 with repetition codes.

Table 4.1 shows some parameters suiting the security constraints (*cf.* Assumptions 1 and 4), for $c_{\mathsf{hard}} = 80$. When a high security is not required, and a somewhat high error probability can be tolerated, much more practical parameters may be obtained.

4.3.2 Experimental results

Table 4.2 shows performances for the repetition codes scenarios described in Table 4.1, based on the implementation of TCHO. Encryption time is roughly equal to the time needed to compute $\mathcal{S}^{\ell}_{\mathcal{L}_P(r_1)}$ (in all scenarios $\mathcal{S}^{\ell}_{\gamma}$ is computed in less than 1 ms), while for decryption the most expensive op-

[1]He then develops this thought: *"When we stated the single law of chance, "events whose probability is sufficiently small never occur", we did not conceal the lack of precision of the statement. There are cases where no doubt is possible; such is that of the complete works of Goethe being reproduced by a typist who does not know German and is typing at random. Between this somewhat extreme case and ones in which the probabilities are very small but nevertheless such that the occurrence of the corresponding event is not incredible, there are many intermediate cases. We shall attempt to determine as precisely as possible which values of probability must be regarded as negligible under certain circumstances. It is evident that the requirements with respect to the degree of certainty imposed on the single law of chance will vary depending on whether we deal with scientific certainty or with the certainty which suffices in a given circumstance of everyday life."* (Chapter 3, *Ibid.*)

eration is the multiplication by K (majority decoding and product by the precomputed matrix always require less than 1 ms). We give average times for a key generation, and the average number of polynomials factorized (trials) during the procedure. The theoretical error probability ρ was accurately verified experimentally, and so is not repeated here.

	encryption	decryption	key gen.	trials
I	29 ms	73 ms	330 s	12
II	37 ms	53 ms	360 s	6
III	42 ms	47 ms	360 s	3
IV	56 ms	170 ms	213 s	2
V	80 ms	215 ms	805 s	10
VI	55 ms	70 ms	682 s	4

Table 4.2: Performances of TCHO2 with repetition codes.

Results in Table 4.2 show that, while selecting parameters, a trade-off must be made between key generation time, encryption and decryption time, and ciphertext expansion. Indeed we cannot obtain both a fast key generation, a low error probability, and fast encryption/decryption; the prohibitive time required by a key generation can be reduced by using larger degree ranges, thereby increasing ρ and the time of encryption and decryption, whilst a small degree range allows better success probability but dramatically slows down key generation. Increasing ℓ reduces ρ but induces a huge expansion and a high time of encryption and decryption. We review the pros and cons of each 128 bits scenario proposed:

III A large interval $[d_{\min}, d_{\max}]$ is chosen, so the key generation time is reduced, but ρ is high.

IV Compared to III, d is reduced, thus key generation is faster, but ρ is higher.

V Here we use a small interval and a larger ℓ, to reach a much lower failure probability, but key generation becomes much slower.

VI A high degree is chosen for K, it allows to reduce its weight, and the length ℓ of a ciphertext, but key generation is still long.

So far our software implementation of TCHO2 is much slower than optimized ones of cryptosystems like NTRU [Sho05], RSA-OAEP [Wal98], or elliptic curves-based systems [Gra06], but may perform much better on a dedicated ASIC, since no complex arithmetic is required, and both LFSR and pseudo-random generators are known to be very fast in hardware (an LFSR implementation in hardware requires about as many gates than the register's length, and outputs one bit per clock cycle).

4.4 Asymptotic parameters

Here we show that secure parameters can be expressed only in terms of the dimension k and c_{hard}, by giving expressions involving constant values in the constraints formulas. For instance, set $w = c_{\mathsf{hard}}$, $d = {c_{\mathsf{hard}}}^2 k$, $\ell = \alpha_1 d$, $d_{\min} = {c_{\mathsf{hard}}}^2$, $d_{\max} = \alpha_2 {c_{\mathsf{hard}}}^2$, and $\gamma = 1 - \beta / c_{\mathsf{hard}}$. The security constraints are satisfied provided that $\alpha_1 > 1$, $\alpha_2 > 1$, and $\beta \geq \ln 4$, with the constraint that $k = \mathcal{O}(c_{\mathsf{hard}})$ and k and c_{hard} are large enough.

Indeed, the constraint on the hardness of LWPM, $w \log \frac{d}{d_{\max}} \geq c_{\mathsf{hard}}$, can be rewritten $k \geq 2\alpha_2$ which always holds for a reasonnable α_2. We also need

$$\gamma \leq 2^{1 - \frac{c_{\mathsf{hard}}}{d_{\min}}}, \text{ that is, } 1 - \frac{\beta}{c_{\mathsf{hard}}} \leq 1 - 2 \ln 2 \frac{c_{\mathsf{hard}}}{{c_{\mathsf{hard}}}^2}$$

and so we need $\beta \geq \ln 4$. Some routine but tedious calculus shows that the failure probability ρ is asymptotically bounded with such parameters. However the parameters thus obtained for TCHO2 may not be practical for small values of k and c_{hard}, since not tight with the security constraints. With the parameters above, key generation runs in time $\mathcal{O}\left({c_{\mathsf{hard}}}^4 k^2\right)$, encryption in $\mathcal{O}\left({c_{\mathsf{hard}}}^2 k\right)$, and decryption in $\mathcal{O}\left({c_{\mathsf{hard}}}^3 k + k^2\right)$, for parameters providing semantic security against adversaries running in time less than $\mathcal{O}\left(2^{c_{\mathsf{hard}}}\right)$.

In comparison, RSA with modulus of k bits offers key generation in time $\mathcal{O}\left(k^4\right)$, encryption in $\mathcal{O}\left(k^2\right)$ decryption in $\mathcal{O}\left(k^3\right)$, and OW-CPA security (namely, the infeasibility to factorize the modulus) against adversaries running in time

$$\mathcal{O}\left(e^{(\frac{64}{9}k)^{1/3}(\ln k)^{2/3}}\right)$$

(GNFS complexity), and so $2^{c_{\mathsf{hard}}}$ security holds with $c_{\mathsf{hard}} = \mathcal{O}\left(k^{1/3}\right)$, whereas in TCHO2 the block size and the security level are almost independent parameters (we only need $k = \mathcal{O}(c_{\mathsf{hard}})$).

4.5 Comparison with other cryptosystems

Although software performances of with our implementation are clearly worse than other asymmetric cryptosystems', TCHO2 may be much more competitive in hardware. Indeed, hardware implementation of RSA [RSA78] is much more complex [Koc95], for example it requires Montgomery method to reduce the number of modular reduction, which is also non-trivial to implement. NTRU [HPS98] also requires modular reductions, and uses a special kind of product between two polynomials with integer coefficients, whereas TCHO2 only works over $\mathbb{F}_2$, much more hardware-friendly. Elliptic curve based systems implementation is also non-trivial (it works on a large finite field). Another singularity of TCHO2 is the independence between a ciphertext length and the security (). which contrast with RSA, NTRU, McEliece [ME78], and GGH [GGH97] for example.

One may notice that TCHO2 looks like McEliece: encryption is "encode and add noise", decryption is "reduce noise and decode". Like TCHO2 it involves a matrix product, a precomputed inversion of matrices related to the private key. McEliece is based on Goppa codes instead of LFSR codes, and mostly relies on the NP-hardness of the problem of decoding an arbitrary linear code. But it suffers from a huge public key (typically 2^{19} bits for secure parameters, whereas TCHO2's is about 2^{13}). Since majority decoding is much more simple than decoding Goppa codes, we are convinced that TCHO2 is more appropriate than McEliece. In addition, both public and private key are much smaller than in McEliece, and decryption requires one matrix-vector product instead of two. Our huge ciphertext expansion is clearly a drawback, but may be acceptable when ciphertexts are not to be kept in memory, and the sole purpose is to encrypt secret keys of a symmetric scheme.

4.6 Conclusion

This variant of TCHO with repetition codes is much more efficient: encryption and decryption algorithms are faster, larger blocks can be encrypted, a precise estimate of the decryption failure probability is given, and experimental results are much better than for TCHO. Besides of that, a huge expansion is required to reach both a negligible error probability and an assumed 2^{80} security (assuming that choosing $c_{\mathsf{hard}} = 80$ is reasonnable today).

Eventually, the bitstream $\mathcal{S}_{\mathcal{L}_P} + \mathcal{S}_\gamma$ can be regarded as trapdoor pseudo-random generator, where the trap allows to reduce the noise enough in order to decode the noised codeword, Other generators of this kind would make it possible to use other codes (not only linear ones), if the use of the trap does not alter the noised pattern. The Blum-Goldwasser [BG85] cipher is an example of trapdoor PRG, where the trap allows to recover the seed of the generator, and thus the entirely cancel the pseudo-random bitstream.

Chapter 5

Security

In this chapter we prove semantic security of TCHO and TCHO2, and design two hybrid encryption schemes offering IND-CCA security.

5.1 One-wayness and non-malleability

Let's begin with the weakest security level:

Proposition 6. *TCHO2 is* $(2^{c_{\mathsf{hard}}}, 2^{-c_{\mathsf{hard}}})$*-OW-CPA secure.*

Proof. It directly follows from the security assumptions 1, and 2 that a plaintext cannot be recovered with probability greater than $2^{-c_{\mathsf{hard}}}$ in time less than $2^{c_{\mathsf{hard}}}$. Hence TCHO2 is $(2^{c_{\mathsf{hard}}}, 2^{-c_{\mathsf{hard}}})$-OW-CPA secure. □

We now state two negative results on TCHO2 security:

Proposition 7. *TCHO is not* $(\mathcal{O}(\ell), 1-\varepsilon)$*-OW-CCA secure, for some small* $\varepsilon > 0$.

Proof. Given a sound ciphertext, it suffices to modify one bit and ask an oracle to decrypt it to get with high probability the plaintext corresponding to the original ciphertext, thus the algorithm runs in constant time, with exactly one query to the oracle. The positive value ε is the probability that a ciphertext of some random message is not sound, that should be small for well chosen parameters. □

As a consequence, it is not IND-CCA secure either, nor NM-CCA secure.

Proposition 8. *TCHO2 is not* $(\mathcal{O}(\ell), 1)$*-NM-CPA secure.*

Proof. If y is a sound ciphertext of x, then $y+x'||\ldots||x'$ is a sound ciphertext of $x + x'$, for any $x' \in \{0,1\}^k$, with probability 1, thus TCHO2 is malleable in constant time, without any encryption query. □

Also remark the property that the sum of n sound ciphertexts is a sound ciphertext, with the same parameters except the bias now equal to γ^w. However the obtain ciphertext shall be impossible to decrypt, even if $n = 2$, for well chosen parameters.

We can define a non-strict notion of *sound ciphertext* for a given key: at decryption, when performing MJD, if the average proportion of correct bits for all the offset does not match with the bias γ (for a random bitstring we get in average as many zeros as ones for a given offset), then with high probability this is not well constructed ciphertext. However, independently of a key pair, any bitstring of length ℓ may be a valid ciphertext. Recall that we talk abound *sound* ciphertexts instead of *valid* ones, since the latter adjective is commonly used for objects that could not have beed produced by the encryption algorithm.

5.2 Semantic security

The results in this section are stated for TCHO2, but hold for TCHO as well.

5.2.1 A sufficient condition

Theorem 7. *If $\mathcal{S}^\ell_{\mathcal{L}_P} + \mathcal{S}^\ell_\gamma$ cannot be distinguished from $\mathcal{S}^\ell_0$ in time t with an advantage larger than ε, then there exists μ such that TCHO2 is $(t - \mu, \varepsilon)$-IND-CPA secure.*

Proof. We proceed by reduction: let $\mathcal{A}^{\mathsf{ror}} = (\mathcal{A}^{\mathsf{ror}}_1, \mathcal{A}^{\mathsf{ror}}_2)$ be an adversary in a real-or-random game, which, given a chosen plaintext $x = \mathcal{A}^{\mathsf{ror}}_1(1^k)$ and a bitstring z of length ℓ, decides whether z is a ciphertext of x or of an unknown randomly chosen plaintext x'; this adversary returns $\mathcal{A}^{\mathsf{ror}}_2(z) \in \{0, 1\}$, and succeeds with an advantage ε, in time t. Since a ciphertext of TCHO2 consists of some bitstring noised with a random source, the ciphertexts space is equal to $\{0, 1\}^\ell$, so there are no trivial instances of the problem, and every element of $\{0, 1\}^\ell$ can be a ciphertext of one or several messages.

We build an adversary against the problem of distinguishing $\mathcal{S}^\ell_{\mathcal{L}_P} + \mathcal{S}^\ell_\gamma$ from $\mathcal{S}^\ell_0$ in the following way: given an unknown instance $\mathcal{S}^\ell_\star$, choose a plaintext $x = \mathcal{A}^{\mathsf{ror}}_1(1^k)$ independently of $\mathcal{S}^\ell_\star$, and compute $z = C(x) + \mathcal{S}^\ell_\star$, then return $\mathcal{A}^{\mathsf{ror}}_2(z)$. If $\mathcal{S}^\ell_\star$ is random, then so is z, otherwise z is a sound ciphertext of x, therefore we got an adversary distinguishing a noised LFSR stream from random with exactly the same advantage than a real-or-random one, in time greater than t. As real-or-random security implies [BDJR97] with no loss semantic security, TCHO2 is IND-CPA secure unless a significant advantage can be obtained on the above problem. □

5.2.2 Distinguishing a noisy LFSR from random

Let P be a random polynomial, such that $\deg(P) \leq \ell$. In order to determine whether a bitstring is $\mathcal{S}^\ell_{\mathcal{L}_P} + \mathcal{S}^\ell_\gamma$ or $\mathcal{S}^\ell_0$, one can try to decode it (*i.e.* recover the initial state of $\mathcal{L}_P$). It is impossible (*cf.* Assumption 2) when $d_P \geq 2c_{\mathsf{hard}}$ and $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$. Another strategy consist in multiplying the stream by P, and deciding whether the obtained stream has bias γ^{w_P} or not. It is impossible to distinguish a random source with bias γ^{w_P} from a uniform one as soon as $\gamma^{w_P} < 2^{-c_{\mathsf{hard}}/2}$. Instead of multiplying by P, one can multiply by multiples of P of lower weight and degree less than ℓ and exploit the obtained bits[1]. For a random P there are in average $\binom{t-2}{v-2}2^{-d_P}$ multiples of weight v and degree t with non-zero constant term, each multiple requiring at least $(\ell - t)v$ operations. Hence the total number of bits of bias γ^v one can obtain using all the multiples of weight v is approximately (for the worst P)

$$N_v \approx 2^{-d_{\max}} \sum_{t=v}^{\ell} (\ell - t) \binom{t-2}{v-2} \approx 2^{-d_{\max}} \binom{\ell - 1}{v}. \tag{5.1}$$

The cost of finding these N_v bits can be lower-bounded by vN_v. If γ^v is small, the advantage of the best distinguisher is [BSW89]

$$\mathsf{Adv} \approx \gamma^v \sqrt{N_v/(2\pi)}.$$

Now, a distinguishing attack will be possible if the complexity vN_v required to obtain N_v bits giving an advantage Adv of 1 is smaller than $2^{c_{\mathsf{hard}}}$. The value of v for which $\mathsf{Adv} = 1$ is

$$v = \frac{d_{\max}}{\log(\ell \gamma^2 e) - \log d_{\max}}. \tag{5.2}$$

It leads to a new assumption.

Assumption 4. *If $d_P \geq 2c_{\mathsf{hard}}$, $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$, and $vN_v > 2^{c_{\mathsf{hard}}}$, where N_v and v are given by equations* (5.1) *and* (5.2)*, then $\mathcal{S}^\ell_{\mathcal{L}_P} + \mathcal{S}^\ell_\gamma$ cannot be distinguished from $\mathcal{S}^\ell_0$.*

Note that the examples of parameters given in Table 4.1 satisfy this constraint. We deduce the following result.

Theorem 8. *Under Assumptions 1 and 4, TCHO2 is $(2^{c_{\mathsf{hard}}}, 2^{-c_{\mathsf{hard}}})$-IND-CPA secure.*

5.3 Hybrid encryption IND-CCA secure

In [FV06] the classical Fujisaki-Okamoto construction [FO99] is applyed to TCHO. Here we propose to build an IND-CCA secure scheme based on

[1] The same idea was used in Section 2.1.3 to compute $\mathcal{I}$.

TCHO2 using two generic hybric constructions, with different requirements. Roughly, the basic KEM/DEM needs a stronger encryption scheme and more random bits than the Fujisaki-Okamoto variant, but the latter requires two random oracles instead of one, and the message has to be encrypted before encapsulating the key.

5.3.1 KEM/DEM

Here the generic KEM/DEM construction [CS04] is used to build an IND-CCA secure scheme. Under Assumptions 1 and 4, TCHO2 is OW-CPA secure [FV06]. It is known [Den02] that a OW-CPA secure asymmetric scheme leads to a IND-CCA secure KEM, so it allows us to build a IND-CCA hybrid encryption scheme with the generic KEM/DEM construction [CS04], using Sym, a symmetric cipher that guarantees indistinguishability under non-adaptive chosen plaintext and ciphertext attacks, and a random oracle H:

Encryption. Given a message x:

1. Choose uniformly a random σ in $\{0,1\}^k$, and a random bitstring r of sufficient length.
2. Compute the symmetric key: $\psi \leftarrow H(\sigma)$.
3. Encapsulate the key: $\chi \leftarrow \mathrm{TCHO2}_{\mathsf{enc}}(\sigma, r)$.
4. Encrypt the message x: $y \leftarrow \mathsf{Sym}_{\mathsf{enc}(\psi)}(x)$.
5. Output the ciphertext (χ, y).

Decryption. Given a ciphertext (χ, y):

1. Compute the encapsulated key: $\psi \leftarrow H(\mathrm{TCHO2}_{\mathsf{dec}}(\chi))$.
2. Decrypt the message: $x \leftarrow \mathsf{Sym}_{\mathsf{dec}(\psi)}(y)$.
3. Output the plaintext x.

5.3.2 Fujisaki-Okamoto revisited

In [AGK05, AGKS05] the Fujisaki-Okamoto construction is converted to a tag-KEM/DEM framework. The encryption scheme obtained offers IND-CCA security when the public encryption scheme is OW-CPA and Γ-uniform (see definition in [FO99]), and the symmetric cipher one-time secure (OW). For instance, one can simply choose $\mathsf{Sym}_{\mathsf{enc}(\psi)}(x) = x + F(\psi)$ for some random oracle F, but Sym can be either a stream cipher or a block cipher.

The construction requires two random oracles H and G. The IND-CPA security of TCHO2 implies OW-CPA security, and the proof of Γ-uniformity of TCHO [FV06] applies to TCHO2 as well. So the following hybrid encryption scheme is IND-CCA secure.

Encryption. Given a message x:

1. Choose a random σ uniformly in $\{0,1\}^k$.
2. Compute the symmetric key: $\psi \leftarrow G(\sigma)$.
3. Encrypt the message x: $y \leftarrow \mathsf{Sym}_{\mathsf{enc}(\psi)}(x)$.
4. Encapsulate the key: $\chi \leftarrow \mathrm{TCHO2}_{\mathsf{enc}}(\sigma, H(\sigma||y))$.
5. Output the ciphertext (χ, y).

Decryption. Given a ciphertext (χ, y):

1. Compute the encapsulated key: $\psi \leftarrow G(\mathrm{TCHO2}_{\mathsf{dec}}(\chi))$.
2. Decrypt the message: $x \leftarrow \mathsf{Sym}_{\mathsf{dec}(\psi)}(y)$.
3. Output the plaintext x.

5.3.3 Practical concerns

Like for a KEM/DEM, only the key of the symmetric scheme is encrypted with TCHO2, and so parameters shall be chosen in function of the key length. Table 4.1 shows example of parameters for a key of 128 bits, a typical length for symmetric schemes. So the two constructions encrypt a message with an overhead of as many bits as in a ciphertext of TCHO2.

On a 4 MHz processor (0.25 μs cycle time), a message is encrypted using an hybrid construction with an overhead of ℓ bits, which is computed in less than 15 ms for $\ell = 50\,000$, when a fast source of random bits is available. The additional cost of the symmetric encryption shall not be an obstacle, and decryption should also be very fast for repetition codes, since it only consists of some simple bitwise operations, and of the counting of the bits in the truncated codeword.

In our software implementation, we may use as symmetric cipher the PRG ISAAC, already used by the generator of biased pseudo-random bits, with as symmetric key a seed on 128 bits.

Chapter 6

Derived constructions

We first present a variant of TCHO2 over a larger finite field, then two other variants, one reducing the expansion but not semantically secure, and one achieving indistinguishability of ciphertexts against some chosen-ciphertext adversaries, called ICCA.

6.1 TCHO2 over $\mathbb{F}_q$

6.1.1 Description

Here $K, P \in \mathbb{F}_q[X]$, LFSR register elements and output are elements of $\mathbb{F}_q$. Again, K has degree d and weight w, P has degree $d_P \in [d_m, d_M]$.

A plaintext is now an element of $\mathbb{F}_q^k$, where k is the dimension of the repetition code.

$\mathcal{S}_\gamma$ is redefined: it produces a stream of elements of $\mathbb{F}_q$; 0 with probability p_γ, otherwise a random element of $\mathbb{F}_q$, so each $b \in \mathbb{F}_q^\star$ appears with probability $(1-p_\gamma)/q$, thus 0 effectively appears with probability $p_\gamma + (1-p_\gamma)/q$.

We still have $K \otimes S_{\mathcal{L}_P} = 0$, and $K \otimes (S_{\mathcal{L}_P} + \mathcal{S}_\gamma) \approx \mathcal{S}_{\gamma^w}$. We will note $p = p_{\gamma^w}$ hereafter.

At decryption, we obtain

$$K \otimes (S_{\mathcal{L}_P} + \mathcal{S}_\gamma + x||x|| \dots ||x) = \mathcal{S}_{\gamma^w} + x'|| \dots ||x'$$

for some $x' \in \mathbb{F}_q^k$. As usual, x is repeated $m = \ell/k$ times, while x' is repeated $m' = (\ell - d)/k$ times.

The linear application transforming x to x' is defined the same way than on $\mathbb{F}_2$.

6.1.2 Reliability

Consider x'_i, the i-th element of the transformed plaintext x'. It is repeated m' times, the expected number of unnoised elements is $p \cdot m'$. The other elements are noised with elements of $\mathbb{F}_q$ (including 0). So the number of

"clear" elements is $p \cdot m' + \frac{1-p}{q} \cdot m'$ on average, where the term $\frac{1-p}{q} \cdot m'$ can be neglected for high enough values of γ and q (*e.g.* when $\gamma = 0.985$, $w = 80$, and $q = 256$).

Unlike on $\mathbb{F}_2$, we do not require absolute majority of clear elements, hence we can allow an error probability greater than $1/2$.

By considering an isolated repetition of x'_i, let $(n_j)_{j=0,\ldots,q}$ be discrete random variables, where n_0 is the number of unnoised elements among the m' repetitions, and n_j, $1 \leq j \leq q$ is the number of elements noised with the j-th element of $\mathbb{F}_q$, for an arbitrary ordering where the first element is 0. The random variable n_0 follows a binomial law with parameters $(m', p + \frac{1-p}{q})$, while each n_j, $1 \leq j \leq q$, follows a binomial law with parameters $(m', \frac{1-p}{q})$.

Let $\mu_j = \Pr[n_0 < n_j]$, the probability that the unnoised bits "lose" again the j-th noise element, that is,

$$\mu_j = \sum_{r=0}^{m'-1} \Pr[n_0 = r] \Pr[n_j > r],$$

and so $\mu_j = \mu_{j'}$, $\forall 1 \leq j \leq j' \leq q$. Let $\mu = \mu_q$. It can also be expressed with a standard normal law as

$$\mu = \varphi\left(-\sqrt{m'}\frac{p}{\sigma}\right)$$

where $\sigma = -p^2 + p + 2\frac{1-p}{q}$, and φ is is the cumulative distribution function of a standard normal distribution:

$$\varphi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt.$$

Thus the probability that the element x'_i is bad decoded is less than

$$(q-1) \cdot \mu.$$

We deduce a bound on the probability of bad decoding of a word composed of k elements $(x'_i)_{i=0,\ldots,k-1}$:

$$\tilde{\rho} \leq 1 - (1 - (q-1)\mu)^k.$$

Basically, we shall decode well the element x'_i when $p \gg \frac{1-p}{q}$, and $p \cdot m' \leq 2$, since we need at least two occurences of the good element to choose it, whilst the probability that a given element of $\mathbb{F}_q^\star$ comes twice is negligible. Moreover, the polynomial P should not allow one to get an advantage on decoding, so we need $\gamma^{d_m/2}$ to be small ($d_m \geq 2 \cdot c_{\mathsf{hard}}$).

The constraints on LWPM remains with this scheme.

Experiments show that the expansion factor does not get better than for TCHO2 on $\mathbb{F}_2$, since the number of bits of a plaintext and a ciphertext are respectively $k \cdot \log q$ and $\ell \cdot \log q$, if q is a power of 2.

Example 11. *We found parameters giving a low error probability for K of degree about 6 000, $\ell = 12\,000$, $q = 2^{32}$ and $k = 8$, so it encrypt 128 bits in 384 000 bits, with a private key on 192 000 bits.*

Eventually, this variant leads to lower values of d and ℓ, but the number of bits of a key and a ciphertext shall increase. Implementation in hardware may be harder, however it may speed up LFSR's processing in software when using extension fields of degree 8 or 32. The number of random bits in terms of ℓ and d_P will increase (to randomly pick elements of $\mathbb{F}_q$), however we shall use smaller ℓ and d_P than in TCHO2.

6.2 A weakly secure scheme with reduced expansion

Assume that the PRG can be seeded with exactly k bits. Let's call this new scheme TCHO3. One encrypts a plaintext x on d_P bits with the following algorithm:

1. set $r \xleftarrow{\$} \{0,1\}^k$,
2. set $y \leftarrow \mathcal{S}_{\mathcal{L}_P(x)} + C(r) + \mathcal{S}_\gamma(r)$,
3. return the ciphertext y.

The decryption algorithm is:

1. recover r from y (usual TCHO2 decryption),
2. compute $\mathcal{S}_{\mathcal{L}_P(x)}$ by eliminating $\mathcal{S}_\gamma(r)$ and $C(r)$,
3. get the initial state x, which is the bitstring formed by the first d_P bits of $\mathcal{S}_{\mathcal{L}_P(x)}$,
4. return the plaintext x.

The decryption procedure is almost the same than in TCHO2, an adversary clearly obtains no more information on the codeword from a ciphertext. Complexities of encryption and decryption do not significatively change.

Like its elder, this new scheme is not OW-CCA secure, for the same reasons. It is also malleable in any adversarial model, since xoring a ciphertext of x with some $\mathcal{S}_{\mathcal{L}_P(x')}$ results in a sound ciphertext of $x + x'$.

We make a new assumption, on the PRG $\mathcal{S}_\gamma$:

Assumption 5. *If the PRG $\mathcal{S}_\gamma$ is seeded on $k \geq c_{\mathsf{hard}}$ bits, then, for random r and r' bitstring of length k, $\mathcal{S}_\gamma(r) + C(r)$ cannot be distinguished from $\mathcal{S}_\gamma(r') + C(r)$ with probability greater than $2^{-c_{\mathsf{hard}}}$ and time less than $2^{c_{\mathsf{hard}}}$.*

Proposition 9. *If $k \geq c_{\mathsf{hard}}$ and the security constraints of TCHO2 are satisfied, then TCHO3 is OW-CPA secure.*

Proof. If $k < c_{\mathsf{hard}}$, an exhaustive search on r could be performed, hence we require $k \geq c_{\mathsf{hard}}$. By Assumption 4, a CPA adversary knowing $\mathcal{S}_{\mathcal{L}_P(x)} + \mathcal{S}_\gamma(r)$ has no information on x. We also assumed that the pseudo-random generator behaved like an ideal one, and so r cannot be recovered either (otherwise we could find $\mathcal{S}_{\mathcal{L}_P(x)}$, that would contradict the assumption).Moreover, we proved (*cf.* Theorem 8) that $\mathcal{S}_{\mathcal{L}_P(x)} + \mathcal{S}_\gamma(r) + C(r')$ does not leak any information on r' to a CPA adversary. Therefore, by Assumption 5, an adversary cannot extract any information on x nor on r from $\mathcal{S}_{\mathcal{L}_P(x)} + \mathcal{S}_\gamma(r) + C(r)$. We deduce that TCHO3 is OW-CPA secure. □

Proposition 10. *TCHO3 is not IND-CPA secure.*

Proof. In an IND game, where x_1 and x_2 are the chosen plaintexts, and y the ciphertext of $x_b, b \in \{0, 1\}$ built by the challenger, an adversary can compute

$$y' = (X^k + 1) \otimes (y + \mathcal{S}_{\mathcal{L}_P(x_1)}).$$

If $b = 1$, then $y' \approx \mathcal{S}_{\gamma^2}$, otherwise it will have bias 0. By computing the weight of y', she thus correctly guess b with high probability, for common parameters: if there are about as many zeros as ones in y', then she returns x_2, otherwise (clearly more zeros than ones), she returns x_1. □

The degree d_P is not fixed, but belongs to an interval $[d_{\min}, d_{\max}]$, thus the length of a message block may depend on the key generation outputs. Against this, we suggest to set $d_{\min}$ to a suitable message length (*e.g.* a multiple of 32), and systematically pad with zeros the remaining bits when $d_P > d_{\min}$.

Compared to the original TCHO2, the ciphertext expansion is clearly reduced: it allows for example to encrypt 5 800 bits instead of 128 in a ciphertext of 50 000 bits (expansion turns from 390 to 8).

6.3 Towards IND security against chosen-ciphertext adversaries

TCHO2 is not OW-CCA, since the attacker can ask for the decryption of the challenge ciphertext modified of only one bit, and recover with high probability the original message. In an adversarial model similar to CCA where the adversary would not be able to oracle-decrypt if the message returned is the challenge's one may prevent this kind of attack: however, since TCHO2 is malleable, one can easily build a ciphertext of $x + x'$ for any known x', and thus recover the challenge message x by querying the oracle for the plaintext $x + x'$. To solve this, we should modify the encryption procedure to introduce

a strict notion of *valid* ciphertext, for example by introducing redundancy proper to the plaintext, so as to make impossible the forgery of a valid ciphertext, of $x + x'$ for example (this is a particular case of malleability, and NM-CCA security is equivalent to IND-CCA security). In the following we define the model ICCA, and show a construction for which IND-ICCA security is reached.

6.3.1 Definitions of ICCA and IPA

We introduces a variant of the CCA model:

Definition 16. *An adversary is called an* irreversible adaptive chosen ciphertext *(ICCA) adversary if she can query the decryption oracle whenever she wants, to decrypt any ciphertext except the challenges, and any other valid ciphertext of their plaintexts. The number of queries and the number of atomic operations must be polynomially bounded.*

This states that the adversary *fails* as soon as she queries for the decryption of a ciphertext whose matching plaintext is already involved in the game: the model gives no trivial way for the adversary to guess whether a ciphertext is a critical one or not; if an attack fails because of the query of a critical ciphertext, this event is not part of the information obtained by the attacker, that is, she does not know that the submitted plaintext indeed encrypts a plaintext of the challenges, but this has *a priori* no effective sense, except if the attacker does not know the rules of the game, or if its memory can be modified, or if she is schizoid. Is the ICCA model really absurd ? If an adversary queries for the decryption of some message, she probably does not know the answer. Meanwhile, the challenger wants her not to know that some messages encrypt some publicly known ciphertext m. Assume she queries for the decryption of a ciphertext of m: the game will end, implicitely saying to the adversary "you should not know what just happened, please forget it", which is indeed absurd. But a concrete way to make this scenario sound would be one where ciphertexts are sent to the oracle, but not remembered by the adversary: when the query is legitimate (the ciphertext does not encrypts m), the oracle would return both the ciphertext and the plaintext, otherwise it would return nothing, assuming that the attacker did not keep any copy of the ciphertext emitted.

We now introduce a particular form of the *plaintext awareness* notion, introduced in [BR94] (see also [BDPR98]), that will help us to prove IND-ICCA security. Informally, an asymmetric cryptosystem is said to be IPA (irreversibly plaintext aware) if it is practically impossible for an adversary to produce a valid ciphertext distinct from the ones already known (*e.g.* given by a challenger) without knowing the matching plaintext, while having access to an encryption oracle (the public key), with the restriction that the adversary should only build ciphertexts for which the matching plaintext is

distinct from all the challenges given in the corresponding game, and from all the plaintexts matching the known ciphertexts. This implicitly states that an observer of the adversary recording every information involved in the construction algorithm should be able to decrypt the ciphertext produced (otherwise the adversary would not know the plaintext, that contradicts the initial postulate). In the classical definition, the restriction stated above does not hold; for instance, RSA is not plaintext aware, since any integer stricly less than the modulus is a valid ciphertext. We now give a more formal definition of the IPA notion:

Definition 17. *Let $\mathcal{A}$ be a Turing machine querying a random oracle, taking as input*

- *pk: a public key chosen by a challenger,*
- *$\mathcal{L}$: a list of ciphertexts of random unknown plaintexts,*

such that both $|\mathcal{L}|$, the number of oracle queries, and the number of atomic operations are in $= \mathcal{O}(\mathcal{P}oly(|pk|))$. This machine outputs a bitstring y, which is a valid ciphertext of some plaintext not encrypted $\mathcal{L}$ (w.r.t. pk*) with probability greater than some $\varepsilon > 0$, over all the $\mathcal{L}$, in time t. We call the machine $\mathcal{A}$ a* (t, ε)-ciphertext creator*.*

An asymmetric encryption scheme is said to be (ε, η)-IPA if and only if, for all $(\mathcal{P}oly(|pk|), \varepsilon')$-ciphertext creator $\mathcal{A}$, with $\varepsilon' \geq \varepsilon$, there exists a deterministic Turing machine $\mathcal{A}^\star$ running in time $\mathcal{P}oly(|pk|)$ – the extractor *– such that, for all y produced by $\mathcal{A}$ with input $\mathcal{L}$ and* pk*,*

$$\Pr[\mathcal{A}^\star(\mathcal{A}, \mathcal{L}, pk) \neq \mathcal{D}(y)] \leq \eta, 0 \geq \eta \geq 1,$$

where $\mathcal{D}$ is the (deterministic) decryption algorithm.

The scheme is simply called IPA if and only if both ε and η are negligible.

The list of ciphertexts $\mathcal{L}$ in the above definition models the capacity of an adversary to eavesdrop a channel. Note that we assume the existence of an extractor, but not that any adversary knows, or can easily find it.

Proposition 11. *If an asymmetric cryptosystem is both (ε, η)-IPA and IND-CPA secure, with ε negligible and η such that*

$$(1 - \frac{1}{\eta})^{2^{c_{\mathsf{hard}}}} \geq 1 - 2^{1-c_{\mathsf{hard}}},$$

then it is IND-ICCA secure.

Proof. In an IND game, an adversary has access to a decryption oracle, but only one valid ciphertext – the challenge – is given, whose query to the oracle is forbidden. Thus the ciphertext creators feeding the ICCA decryption

oracle all have $|\mathcal{L}| = 0$. We show that, for all IND-ICCA adversary with non-negligible success probability and polynomial running time, we can build an IND-CPA adversary with equal running time and still non-negligible success probability.

Consider a (χ, ξ)-IND-ICCA adversary for a (ε, η)-IPA scheme. She makes at most χ decryption queries to the oracle. Using the IPA extractor instead of the decryption oracle leads to a *perfect simulation* with probability greater than

$$(1 - \frac{1}{\eta})^{\chi},$$

that is, the probability that each ciphertext is "decrypted" correctly by the extractor. Note that the negation of this is not even a proved sufficient condition for the failure of the attack, but we will assume it. The IND-CPA adversary built this way hence succeeds with probability greater than

$$\xi - 1 + (1 - \frac{1}{\eta})^{\chi}.$$

If ξ is non negligible, so is this last value, as soon as

$$(1 - \frac{1}{\eta})^{2^{c_{\mathsf{hard}}}} \geq 1 - 2^{1-c_{\mathsf{hard}}}.$$

Thus we built a $(\chi, \xi - 1 + (1 - \frac{1}{\eta})^{\chi})$-IND-CPA adversary from a (χ, ξ)-IND-ICCA adversary, and so is the initial condition on η implies that for all efficient IND-ICCA adversary, the decryption oracle can be replaced by the IPA extractor. By inversion, IND-CPA security implies IND-ICCA security, provided that η verifies the inequality above stated. □

Proposition 12. *Let $\mathcal{V}(\mathsf{pk},\mathsf{sk})$ be the number of valid ciphertexts for the key pair $(\mathsf{pk},\mathsf{sk})$. If an asymmetric scheme is OW-CPA and*

$$\frac{\mathcal{V}(\mathsf{pk},\mathsf{sk})}{2^{\ell} - \mathcal{V}(\mathsf{pk},\mathsf{sk})} \geq \frac{1}{2^{c_{\mathsf{hard}}}},$$

then it is not IPA.

Proof. If the ratio of valid ciphertexts is greater than $1/2^{c_{\mathsf{hard}}}$, then the adversary who randomly picks a bitstring of the same length than a ciphertext obtains a valid ciphertext with probability greater than $1/2^{c_{\mathsf{hard}}}$. In this simple algorithm, the only *information* the adversary has is *this* ciphertext. Hence if the scheme is OW-CPA, an adversary cannot recover the plaintext, and so no polynomial time extractor exists. Finally, there exists an adversary able to compute a valid ciphertext with probability greater than $1/2^{c_{\mathsf{hard}}}$ such that no polynomial time extractor exists, which contradicts the IPA definition. □

6.3.2 Notion of valid ciphertext and IND-ICCA security

Addition of deterministic redundancy

Among the k bits of the codeword, M contain the plaintext, and $R = k - M$ the redundancy, defined by a function

$$\mathfrak{R} : \{0,1\}^M \to \{0,1\}^R.$$

A ciphertext of x is

$$y = \mathcal{S}_{\mathcal{L}_P(r_1)} + C(x||\mathfrak{R}(x)) + \mathcal{S}_\gamma(r_2),$$

for randomly chosen r_1 and r_2. Given a ciphertext y, decryption is performed with the following algorithm:

1. recover the coded word $x||r$,
2. if $\mathfrak{R}(x) = r$, return the plaintext x,
3. return $\perp$ otherwise.

Assume that $\mathfrak{R}$ is a random injection: in an IND scenario, given x_1 and x_2, along with the ciphertext c of one of those, the adversary can ask for the decryption of $c + y||\mathfrak{R}(x_1 + y) + \mathfrak{R}(x_1)||\ldots$; if the oracle answers $x_1 + y$, then she returns x_1, otherwise (for answer $\perp$ or $z \in \{0,1\}^M$), she returns x_2. Hence no matter how "good" is $\mathfrak{R}$, IND-ICCA security will never be achieved.

Addition of non-deterministic redundancy

Now the codeword on k bits contains the plaintext on M bits, followed by N random bits picked by the encrypter, and finally $R = k - M - N$ bits for to the image of the function $\mathfrak{R}$, which now takes two arguments, x and $\aleph$, the latter being the bitstring of N random bits ($\aleph$ must be included in the codeword in order to check a ciphertext's validity). We define

$$\mathfrak{R} : \{0,1\}^M \times \{0,1\}^N \to \{0,1\}^R.$$

A ciphertext of x is

$$y = \mathcal{S}_{\mathcal{L}_P(r_1)} + C(x||\aleph||\mathfrak{R}(x)) + \mathcal{S}_\gamma(r_2).$$

The decryption algorithm of y is:

1. recover the coded word $x||\aleph||r$,
2. if $r = \mathfrak{R}(x, \aleph)$, return the plaintext x,
3. return $\perp$ otherwise.

The decryption oracle associated would return the plaintext x, but not $\aleph$ (nor $\mathfrak{R}(x, \aleph)$). From now we consider $\mathfrak{R}$ as a random oracle, and will call the encryption scheme TCHO4.

We recall that, as TCHO2, the system created with non-deterministic redundancy cannot be IND-CCA, since, in a OW game, a query to the decryption oracle with the challenge ciphertext with only one bit modified would return with high probability the encrypted message. Since the basic TCHO2 is IND-CPA secure, it trivially still holds with the non-deterministic redundancy above described (simply replace x in TCHO2 by $x||\aleph||\mathfrak{R}(x, \aleph)$).

Note the following fact:

$$\aleph \text{ recovered} \Rightarrow x \text{ recovered},$$

This implication is trivial (*e.g.* XOR the ciphertext with $x||0||(\mathfrak{R}(x, \aleph) + \mathfrak{R}(0, \aleph))||\ldots$ and look for 0 as decrypted message). However the converse is not so obvious; if we had "x recovered $\Rightarrow$ $\aleph$ recovered", an ICCA adversary would win an IND game trivially: assuming that the plaintext encrypted is the first of the two challenges plaintexts, one recovers $\aleph$, if she is wrong (she can check it by applying the strategy to recover x from $\aleph$), she returns the other challenge plaintext, and the one chosen otherwise. Therefore this implication is sufficient to win the IND game, but maybe not necessary, since finding $\aleph$ is not formally required. Thus we cannot reduce our problem to the computation of $\aleph$ from x given a ciphertext of x.

Let's consider $S(x)$, the set of all ciphertexts build by using $\aleph$ as non-deterministic seed, and ξ in place of $\mathfrak{R}(x, \aleph)$. We can define a binary equivalence relation over $S(x)$ such that two elements are equivalent if and only if they were built with the same ξ. Therefore $S(x)$ can be partitioned into 2^R subsets of equal size matching the equivalence classes defined by this relation. Among those classes, only one contains valid ciphertexts (and only valid ones): the one where $\xi = \mathfrak{R}(x, \aleph)$. Hence if an adversary has no information on $\mathfrak{R}(x, \aleph)$, there is no way to choose the right class with a significant advantage. In particular, a triplet $(x, \aleph, \mathfrak{R}(x, \aleph))$ cannot be distinguished from a triplet $(x, \aleph, \psi), \psi \in \{0, 1\}^R$, without querying for $\mathfrak{R}(x, \aleph)$. This argument is used to prove the following theorem.

Theorem 9. *If the constraints required for the semantic security of TCHO2 translated to TCHO4 are satisfied, and if $\mathfrak{R}$ is a random oracle, then TCHO4 is $(2^{-R}, 1)$-IPA.*

Proof. Consider an IPA adversary $\mathcal{A}$; by querying the oracle $\mathfrak{R}$, she gets triplets $(x_i, \aleph_i, \mathfrak{R}(x_i, \aleph_i))$, $i = 1, \ldots, L$. Using pk, she also obtains pairs (x_i, y_i), $i = L + 1, \ldots, M$, with $M = \mathcal{O}(|\mathsf{pk}|^\alpha)$. Let y be the bitstring returned by $\mathcal{A}$. When $\mathcal{D}(y) = \perp$ or $\mathcal{D}(y) \notin \{x_1, \ldots, x_M\}$, let's denote W the bitstring encoded in $C(W)$, decomposed in three subbitstrings $W = x||\aleph||r$. We can distinguish two cases:

1. $x \notin \{x_1, \ldots, x_M\}$: then $\mathcal{A}$ succeeds as soon as $r = \mathfrak{R}(x, \aleph)$. We have $\Pr[r = \mathfrak{R}(x, \aleph)] \leq 2^{-R}$, since $\mathfrak{R}$ is a random oracle.

2. $r \neq \mathfrak{R}(x, \aleph)$: then $\mathcal{A}$ fails with probability 1, since W cannot be a valid ciphertext.

By definition of IPA, x cannot be in $\{x_1, \ldots, x_M\}$, so this is the only cases we consider. Finally, $\Pr[\mathcal{A} \text{ succeeds}] \leq 2^{-R}$. Hence any $(\mathcal{P}oly(\mathsf{pk}), \varepsilon)$-ciphertext creator with $\varepsilon > 2^{-R}$ needs to query for $\mathfrak{R}(x, \aleph)$, and so the extractor succeeds with probability strictly greater than $1 - 2^{-R}$, since it reads the oracle queries of the creator. It proves that TCHO4 is $(2^{-R}, 1)$-IPA. □

Theorem 10. *If* $(1 - 2^{-R})^{2^{c_{\mathsf{hard}}}} \geq 1 - 2^{-c_{\mathsf{hard}}}$, *then TCHO4 is IND-ICCA secure.*

Proof. We know that TCHO4 is IND-CPA. By Proposition 11 and Theorem 9, the result follows. □

If $\mathfrak{R}$ is not a random oracle but a given function, it has to satisfy several properties. If $\mathfrak{R}$ is linear on $\aleph$ (*i.e.* $\mathfrak{R}(x, \aleph) + \mathfrak{R}(x, \aleph') = \mathfrak{R}(x, \aleph + \aleph')$), then in an IND game, if x is one of the two plaintexts challenges, one only has to XOR the ciphertext with $0||\aleph'||\mathfrak{R}(x, \aleph')||\ldots$ and query the decryption oracle: if it answers x, then the adversary returns x, otherwise she returns the second plaintext. Hence $\mathfrak{R}$ must be non-linear on $\aleph$. Moreover, we require that the number "linear pairs" is small, that is,

$$\max_x \frac{\#\{(\aleph, \aleph'), \aleph \neq \aleph', \mathfrak{R}(x, \aleph) + \mathfrak{R}(x, \aleph') \neq \mathfrak{R}(x, \aleph + \aleph')\}}{2^N \cdot (2^N - 1)} \leq \frac{1}{2^{c_{\mathsf{hard}}}}.$$

If we XOR the challenge ciphertext with $y||0||\mathfrak{R}(y, \aleph')||\ldots$, one may obtain $(x + y)||\aleph||(\mathfrak{R}(x, \aleph) + \mathfrak{R}(y, \aleph'))$ in certain cases. To prevent this, we require

$$\max_{y, \aleph'} \Pr_{x, \aleph} \left[\mathfrak{R}(x, \aleph) + \mathfrak{R}(y, \aleph') = \mathfrak{R}(x + y, \aleph)\right] \leq \frac{1}{2^{c_{\mathsf{hard}}}}.$$

The attack against IND-ICCA security mentionned in the previous section is infeasible as soon as $N > c_{\mathsf{hard}}$ and

$$\Pr_{\aleph \neq \aleph'}[\mathfrak{R}(x, \aleph) = \mathfrak{R}(x, \aleph')] \leq \frac{1}{2^{c_{\mathsf{hard}}}},$$

considering an exhaustive search on the random bitstring $\aleph$.

For instance, if $N = c_{\mathsf{hard}} + 1$, we require $R \geq N$; so we need $k > 2c_{\mathsf{hard}} + M$, *e.g.* to encrypt 32 bits with 2^{80} security k should be greater than 192. It induces a ciphertext length of about 60 000 bits.

Conclusion

We studied the existing probabilistic encryption scheme TCHO, implemented and improved it; we designed efficient algorithms for the generation of a biased random bitstring and a large LFSR output, including an almost optimal one for the former. The new cryptosystem TCHO2 leads to a slightly faster encryption, and an exponentially faster decryption, while introducing new security constraints and obviating the need for a primitive polynomial as public key. We proved, under certain assumptions, that both TCHO and TCHO2 could achieve semantic security, and suggest two known hybrid schemes to reach the strongest level of security, namely IND-CCA security. We also suggest several variants of our scheme, either sacrificing semantic security to get a low expansion, or reaching IND-ICCA security at the cost of a huger expansion.

Applications may be found in embedded environments, to provide a simple encryption procedure. Passive RFID tags may also find with TCHO2 a way to use public key cryptography, actually infeasible with other asymmetric primitives on their small architectures; this may solve important problems of privacy in RFID protocols. The expansion would "only" result in an overhead of about 5 Kb in an hybrid framework. Moreover, unlike RSA, TCHO2 would not be harmed by a quantum computer, since no feasible quantum algorithm is known to solve the problems it relies on (this kind of cryptosystem is sometimes called *post-quantum*).

Finally, as TCHO2 security only relies on heuristic assumptions, further work could be devoted to giving concrete elements of proof, *e.g.* concerning the problem LWPM, or finding other models of trapdoor pseudo-random generators exploiting the error correction capacity of certain codes.

Bibliography

[AGK05] Masayuki Abe, Rosario Gennaro, and Kaoru Kurosawa. Tag-KEM/DEM: A new framework for hybrid encryption. IACR ePrint archive 2005/027, 2005. Available at `http://eprint.iacr.org/2005/027`. Newer version in [AGKS05].

[AGKS05] Masayuki Abe, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa-Desmedt KEM. In *EUROCRYPT'05*, pages 128–146, 2005. Older version in [AGK05].

[AM94] Leonard Adleman and Kevin McCurley. Open problems in number theoretic complexity, II. In L. Adleman and M.-D. Huang, editors, *ANTS-I*, volume 877 of *Lecture Notes in Computer Science*, pages 291–322. Springer, 1994.

[Arn05] Jörg Arndt. *Algorithms for programmers*. Available at `http://www.jjj.de/fxt/`, 2005.

[BBS86] Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15:364–383, 1986.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS'97*, page 394. IEEE Computer Society, 1997.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO'98*, pages 26–45. Springer, 1998.

[Ber68] Elwyn R. Berlekamp. *Algebraic coding theory*. McGraw-Hill, 1968.

[BG85] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In G. R.

Blakley and D. C. Chaum, editors, *CRYPTO'84*, pages 289–302. Springer, 1985.

[BGP06] Côme Berbain, Henri Gilbert, and Jacques Patarin. Quad: A practical stream cipher with provable security. In *EUROCRYPT*, pages 109–128, 2006.

[BHSV98] Mihir Bellare, Shai Halevi, Amit Sahai, and Salil P. Vadhan. Many-to-one trapdoor functions and their relation to public-key cryptosystems. In *CRYPTO '98*, pages 283–298. Springer, 1998.

[BLZ03] Richard Brent, Samuli Larvala, and Paul Zimmermann. A fast algorithm for testing reducibility of trinomials mod 2 and some new primitive trinomials of degree 3021377. *Mathematics of Computation*, pages 1443–1452, 2003.

[Bor43] Emile Borel. *Les probabilités et la vie.* Presses Universitaires Françaises, 1943.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In A. De Santis, editor, *EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[BSW89] Paul T. Bateman, John L. Selfridge, and Samuel S. Wagstaff. The new mersenne conjecture. *American Mathematical Monthly*, 96(2):125–128, 1989.

[BZ03] Richard Brent and Paul Zimmermann. Algorithms for finding almost irreducible and almost primitive trinomials. Primes and Misdemeanours: Lectures in Honour of the Sixtieth Birthday of Hugh Cowie Williams (edited by A. van der Poorten and A. Stein), 2003.

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[CM01] Sandeepan Chowdhury and Subhamoy Maitra. Efficient software implementation of linear feedback shift registers. In C. Pandu Rangan and C. Ding, editors, *INDOCRYPT'01*, volume 2247 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2001.

[CM03] Sandeepan Chowdhury and Subhamoy Maitra. Efficient software implementation of LFSR and boolean function and its application in nonlinear combiner model. In J. Zhou, M. Yung, and Y. Han, editors, *ACNS'03*, volume 2846 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2003.

[CMI03] Paul Camion, Miodrag J. Mihaljević, and Hideki Imai. Two alerts for design of certain stream ciphers: Trapped LFSR and weak resilient function over GF(q). In K. Nyberg and H. Heys, editors, *SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 2003.

[CS04] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2004.

[CT00] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity check equations of weight 4 and 5. In B. Preneel, editor, *EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer, 2000.

[CZ81a] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[CZ81b] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[Den02] Alexander W. Dent. A designer's guide to KEMs. Public report NES/DOC/RHU/WP5/029/1, NESSIE project, 2002. Available at `http://eprint.iacr.org/2002/174`.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[Ekd03] Patrik Ekdahl. *On LFSR based Stream Ciphers - Analysis and Design*. PhD thesis, Lund University, 2003.

[Ell70] James H. Ellis. The possibility of secure non-secret digital encryption. GCHQ-CESG publication, 1970.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. Wiener, editor, *CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[FV06] Matthieu Finiasz and Serge Vaudenay. TCHo: the trapdoor stream cipher. unpublished, 2006.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO '97*, Lecture Notes in Computer Science, pages 112–131. Springer, 1997.

[Gil64] Donald B. Gillies. Three new mersenne primes and a statistical theory. *Mathematics of Computation*, 18:93–97, 1964.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC'82*, pages 365–377. ACM Press, 1982.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[GM01] Kishan Chand Gupta and Subhamoy Maitra. Multiples of primitive polynomials over GF(2). In C. Pandu Rangan and C. Ding, editors, *INDOCRYPT'01*, volume 2247 of *Lecture Notes in Computer Science*, pages 62–72. Springer, 2001.

[Gol01] Oded Goldreich. *Foundactions of Cryptography*, volume 1. Cambridge University Press, 2001.

[GPR06] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. Analysis of the linux random number generator. Cryptology ePrint Archive, Report 2006/086, 2006. Available at `http://eprint.iacr.org/`.

[Gra06] Torbjörn Granlund. GNU multiple precision arithmetic library (GMP), 2006. Available at `http://swox.com/gmp/`.

[Guy94] Richard K. Guy. *Unsolved problems in number theory*. Springer, 2nd edition, 1994.

[GvP98] Shuhong Gao, Joachim von zur Gathen, and Daniel Panario. Gauss periods: orders and cryptographical applications. *Mathematics of Computation*, 67(221):343–352, 1998.

[HN99] Miia Hermelin and Kaisa Nyberg. Correlation properties of the bluetooth combiner. In *ICISC'99*, Lecture Notes in Computer Science, pages 17–29, 1999.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In J. Buhler, editor, *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[HR17] Godfrey H. Hardy and Srinivasa Ramanujan. The normal number of prime factors of a number n. *The Quarterly Journal of Mathematics*, pages 76–92, 1917. also published in "Collected papers of Ramanujan", Cambridge University Press, 1927.

[HWL+91] D. G. Hoffman, Wal, D. A. Leonard, C. C. Lidner, K. T. Phelps, and C. A. Rodger. *Coding Theory: The Essentials.* Marcel Dekker, Inc., 1991.

[Jam00] K Jambunathan. On choice of connection-polynominals for LFSR-based stream ciphers. In B. K. Roy and E. Okamoto, editors, *INDOCRYPT'00*, volume 1977 of *Lecture Notes in Computer Science*, pages 9–18. Springer, 2000.

[Jen96a] Robert J. Jenkins. ISAAC. In D. Gollmann, editor, *FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 41–49. Springer, 1996.

[Jen96b] Robert J. Jenkins. ISAAC. In D. Gollmann, editor, *FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 41–49. Springer, 1996.

[JJ99] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks based on turbo code techniques. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 1999.

[Koc95] Cetin Kaya Koc. RSA hardware implementation. Technical Report TR801, RSA Laboratories, 1995.

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.

[LP98] Rudolf Lidl and Günter Pilz. *Applied abstract algebra, 2-nd ed.* Springer, 1998.

[Lus06] Peter Luschny. *Fast Factorial Functions*, 2000–2006. `http://www.luschny.de/math/factorial/`.

[LV04a] Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In Matthew K. Franklin, editor, *CRYPTO'04*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer, 2004.

[LV04b] Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In M. K. Franklin, editor,

CRYPTO'04, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer, 2004.

[Mar95a] Georges Marsaglia. *The Diehard Battery of Tests of Randomness*, 1995. Available at `http://stat.fsu.edu/pub/diehard/`.

[Mar95b] Georges Marsaglia. *The Diehard Battery of Tests of Randomness*, 1995. Available at `http://stat.fsu.edu/pub/diehard/`.

[ME78] Robert J. Mc Eliece. A public-key cryptosystem based on algebraic coding theory. Technical report, Jet Propulsion Lab Deep Space Network Progress report, 1978.

[MGV05] Subhamoy Maitra, Kishan Chand Gupta, and Ayineedi Venkateswarlu. Results on multiples of primitive polynomials and their products over GF(2). *Theoretical Computer Science*, 341(1-3):311–343, 2005.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[MNT02] Atsuko Miyaji, Masao Nonaka, and Yoshinori Takii. Known plaintext correlation attack against RC5. In *CT-RSA'02*, Lecture Notes in Computer Science, pages 131–148. Springer, 2002.

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[MS88] Willi Meier and Othmar Staffelbach. Fast correlation attacks on stream ciphers. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 1988.

[MS94] Willi Meier and Othmar Staffelbach. The self-shrinking generator. In A. De Santis, editor, *EUROCRYPT'94*, pages 205–214. Springer, 1994.

[MS01] Itsik Mantin and Adi Shamir. A practical attack on broadcast RC4. In M. Matsui, editor, *FSE'01*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2001.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[Pud01] Marina Pudovkina. A known plaintext attack on the ISAAC keystream generator. IACR ePrint Archive, Report 2001/049, 2001. Available at `http://eprint.iacr.org/2001/049`.

[Rom92] Steven Roman. *Coding and information theory*. Springer, 1992.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSP98] Andreas Rieke, Ahmad-Reza Sadeghi, and Werner Poguntke. On primitivity tests for polynomials. In *ISIT'98*, 1998.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[Sho90] Victor Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Information Processing Letters*, 33(5):261–267, 1990.

[Sho05] Victor Shoup. *NTL: A Library for doing Number Theory*, 2005. Available at `http://shoup.net/ntl/`.

[Sie86] Thomas Siegenthaler. Cryptanalysts representation of nonlinearly filtered ML-sequences. In Franz Pichler, editor, *EUROCRYPT'85*, volume 219 of *Lecture Notes in Computer Science*, pages 103–110. Springer, 1986.

[Ste87] Jacques Stern. Secret linear congruential generators are not cryptographically secure. In *In Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 421–426, 1987.

[TM71] Myron Tribus and Edward C. McIrvine. Energy and information. *Scientific American*, 225(3):179–188, 1971. (Note: the table of contents in this volume incorrectly lists this as volume 224).

[Ver26] Gilbert S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Journal of the IEEE*, pages 109–115, 1926.

[Wag83] Samuel S. Wagstaff. Divisors of mersenne numbers. *Mathematics of Computation*, 40:385–397, 1983.

[Wag02] David Wagner. A generalized birthday problem. In M. Yung, editor, *CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2002.

[Wal98] John Walker. *ENT: A Pseudorandom Number Sequence Test Program*, 1998. Available at `http://www.fourmilab.ch/random/`.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on the Foundations of Computer Science, IEEE*, pages 80–91, 1982.

Appendices

A Weak initial states in ISAAC

We reproduce below a paper written at the end of the internship.

Abstract. In this note, we study the deterministic random bits generator ISAAC. We present more than $2^{8\,135}$ initial states inducing a strongly biased distribution of the bits produced at the first round of the algorithm, and a strong distinguisher requiring 2^{176} samples. We also show 2^{32} states that can be recovered from the firsts 8 192 bits produced in less than 30 seconds with a paper an a pen, and point out minor weaknesses of the algorithm. A modification of the algorithm is proposed to fix some of the flaws presented.

ISAAC is a deterministic random bits generator designed in 1996. Its author claims [Jen96b] that it has "*no bad initial states, not even the state of all zeros*". We investigated the question, and focus in this note on several minor weaknesses and more than $2^{8\,135}$ states. We start by presenting ISAAC, and end with a proposal of a modification of the algorithm.

1 ISAAC

1.1 Presentation

ISAAC is derived from the stream cipher RC4. Although it is *"designed to be cryptographically secure"* [Jen96b], no security proof is given, and it seriously lacks analysis: only statistical tests argue for its security [Jen96b], and until now, only one publication [Pud01] tackled it, presenting a state recovery attack running in time $10^{1\,240}$.

We follow the description of the algorithm provided in Figure 4 of [Jen96b]; the internal state is an array of 256 32-bit words, and at each round, the algorithm outputs another array of 256 32-bit words. In the following, S denotes the initial state, and S_i its ith element, while β denotes the first output, and β_i its ith element, for $i \in \{0, \ldots, 255\}$. The generation algorithm takes as parameters three values a, b and c, the first two are 32-bit, the third is 8-bit,

initialized to an arbitrary value, and modified at each round: a is used as a kind of entropy accumulator, b contains the previous pseudo-random word, and c is a simple counter, incremented at each round of the algorithm. Their initial values are public, and are not part of the secret initial state. We give the generation algorithm in a readable form in Algorithm .1, for an arbitrary round, where the internal state is I, the output is O, and the inputs a, b, and c are those computed in the previous round. The symbol $\oplus$ denotes the bitwise XOR, $+$ stands for the usual integer addition, and $\ll$, $\gg$, are bit shifting operators "à la C".

Input: a, b, c, and the internal state I, an array of 256 32-bit words
Output: an array O of 256 32-bit words
1: $c \rightarrow c + 1$
2: $b \rightarrow b + c$
3: **for** $i = 0, \ldots, 255$ **do**
4: $\quad x \rightarrow I_i$
5: $\quad a \rightarrow f(a, i) + I_{(i+128) \mod 256}$
6: $\quad I_i \rightarrow a + b + I_{(x \gg 2) \mod 256}$
7: $\quad O_i \rightarrow x + S_{(I_i \gg 10) \mod 256}$
8: $\quad b \rightarrow O_i$
9: **end for**
10: **return** O

Algorithm .1: ISAAC algorithm for an arbitrary round.

The value $f(a, i)$ in Algorithm .1 is a 32-bit word, defined for all a and $i \in \{0, \ldots, 255\}$ as:

$$f(a,i) = \begin{cases} a \ll 13 & \text{if } i \equiv 0 \mod 4 \\ a \gg 6 & \text{if } i \equiv 1 \mod 4 \\ a \ll 2 & \text{if } i \equiv 2 \mod 4 \\ a \gg 16 & \text{if } i \equiv 3 \mod 4 \end{cases}.$$

We deduce the algorithm used to compute the first ouput β from the initial state S, depicted in Algorithm .2. This redundant algorithm is given for a better understanding of the following developments.

1.2 Observations

The notation $\equiv$ stands for the equivalence modulo 2^{32} hereafter.

Theorem 1. *For a random initial state S, and fixed a, b, and c,*

$$\Pr\left[\exists i \in \{1, \ldots, 255\}, \beta_0 \equiv S_0 + S_i \mod 2^{32}\right] \geq \frac{255}{256} \approx 0.9961.$$

Input: a, b, c, and the initial state S
Output: the array β of 256 32-bit words
1: $b \rightarrow b + c + 1$
2: **for** $i = 0, \ldots, 255$ **do**
3: $\quad x \rightarrow S_i$
4: $\quad a \rightarrow f(a, i) + S_{(i+128) \mod 256}$
5: $\quad S_i \rightarrow a + b + S_{(x \gg 2) \mod 256}$
6: $\quad \beta_i \rightarrow x + S_{(S_i \gg 10) \mod 256}$
7: $\quad b \rightarrow \beta_i$
8: **end for**
9: **return** β

Algorithm .2: ISAAC algorithm computing the first ouput β from the initial state S.

Proof. Let $\mu = f(a, 0) + S_{128} + b + c + 1 + S_{(S_0 \gg 2) \mod 256}$, be the value obtained at line 5 of Algorithm .2, at the first iteration ($i = 0$). At line 6 , when $i = 0$, we get $\beta_0 = S_0 + \lambda$, where $\lambda = \mu$ if $(\mu \gg 10) \mod 256 \neq 0$, and $\lambda = S_{(\mu \gg 10) \mod 256}$ otherwise. Since S_0 is random, $(S_0 \gg 2) \mod 256$ is a random value in $\{0, \ldots, 255\}$. Since S_{128} is random, then μ is a random value in $\{0, \ldots, 2^{32} - 1\}$. Hence $\mu \gg 10 \mod 256 \neq 0$ with probability $255/256$, which proves the result. □

When there exists $1 < i < 256$ such that $\beta_0 = S_0 + S_i$, S_0 and i are correctly guessed with probability respectively 2^{-32} and $1/255$. Thus one recovers S_0 and S_i for a certain i, with probability $2^{-32} \times 1/255 \times 255/256 = 2^{-40}$, whereas ideally this probability should be 2^{-64}.

Theorem 2. *Let i a given value in $\{0, \ldots, 255\}$. For a random initial state S, and fixed a, b, and c,*

$$\Pr\left[\beta_0 - \beta_1 \equiv S_0 - S_i\right] \geq \frac{254}{256^2} \approx 0.0039.$$

Proof. We distinguish two cases, for random S, and fixed a, b, c:

- $i = 1$: from the previous theorem, we get $\beta_0 \equiv S_0 + S_j$ and $\beta_1 \equiv S_1 + S_j$, for some $1 < j < 256$, with probability $254/256^2$.
- $i \neq 1$: for similar reasons, we get $\beta_0 \equiv S_0 + S_1$ and $\beta_1 \equiv S_1 + S_i$ with probability $254/256^2$.

Eventually, for all fixed i, $\beta_0 - \beta_1$ is equivalent to $S_0 - S_i$ with probability greater than $254/256^2$, which completes the proof. □

Generalizing this to all the couples (β_i, β_j), the average number of collisions (of pairs $(\beta_i = S_e + S_f, \beta_j = S_g + S_f)$, for some e, f, g in $\{0, \ldots, 255\}$)

is

$$\sum_{i=1}^{254} i \times \frac{256-i}{256} \approx 43.$$

Nevertheless, since there is no trivial way to identify the colliding pairs among the $128 \times 255 = 32\,640$ possible ones, the interest of this last result is limited. But note that two previous facts would be dramatic if ISAAC was used as a keystream generator; it would allow a passive adversary to obtain information on the key (namely the equivalence class of $S_0 - S_i$) with probability $254/256^2$.

Theorem 3. *Let $N \in \{1, \ldots, 127\}$, and set $S_i = X$ for all $i > N$, and $S_i = Y$ for $i \leq N$, with fixed positive integers $X < 2^9$ and $Y < 2^{10}$. When $a = b = c = 0$, the following facts arise:*

- *if $N = 0$, then*

$$\beta_0 = \begin{cases} X + 2Y + 1 & \text{if } Y \in \{0, \ldots, 3\} \\ 2X + Y + 1 & \text{if } Y \in \{4, \ldots, 2^{10} - 1\} \end{cases},$$

- *if $N = 1$, then*

$$\beta_0 = \begin{cases} X + 2Y + 1 & \text{if } Y \in \{0, \ldots, 7\} \\ 2X + Y + 1 & \text{if } Y \in \{8, \ldots, 2^{10} - 1\} \end{cases},$$

- *and generally, for $0 \leq N < 128$, if $S_0 = \cdots = S_{N-1} = k$, then*

$$\beta_0 = \begin{cases} X + 2Y + 1 & \text{if } Y \in \{0, \ldots, M\} \\ 2X + Y + 1 & \text{if } Y \in \{M + 1, \ldots, 2^{10} - 1\} \end{cases},$$

  with $M = \max\{m, (m \gg 2) < N\}$.

Proof. These results directly follow from Algorithm .2. and were verified automatically with the original source code [Jen96b] for all (X, Y). □

The limitation of X to 2^9 comes from the fact that above this limit, $(S_i \gg 10) \neq 0$ (*cf.* line 6 of Algorithm .2). We also need $Y < 2^{10}$ so that, at line 5, we do not pick an index less than N, that is, for which $S_i = Y$. For the general case, the limit M comes from the fact that, at the line 5 of Algorithm .2, we shall pick the value Y as soon as $Y \gg 2$ is less than $N - 1$, and X otherwise. Finally, we need $N < 128$ in order to get $i + 128 > N \mod 256$ for all $i \in \{0, \ldots, N - 1\}$ (*cf.* line 4), and so $a = X$. We obtain exactly $2^9 \times 2^{10} \times 2^7 = 2^{26}$ such states.

2 A class of more than $2^{8\,135}$ weak initial states

2.1 Properties

The states considered have a fraction of random elements, and the remaining elements are fixed to the same value.

Set $N \in \{2, \ldots, 256\}$ such that $S_i = X$, for all $i < N$, for a fixed positive $X < 2^{32}$, and the other S_i's are all random 32-bit words. Then, for a random X:

$$\Pr[\beta_i \equiv 2X] \geq \frac{N-1-i}{256}, i = 0, \ldots, N-1.$$

Indeed, at line 6 of Algorithm .2, we have $x = X$, and so $S_{(S_i \gg 10) \mod 256}$ is equal to X if $(S_i \gg 10) \mod 256$ is greater than i and strictly less than N, that occurs with probability about $(N-1-i)/256$, by Theorem 1 The inequality comes from the fact that, if we pick an index less than i, the word at this position is X with probability 2^{-32}. Eventually the value $2X$ shall appear with high probability, compared to a random bitstream. There are approximately $2^{32\times 254} = 2^{8\,128}$ such weak states.

For example, if $N = 64$ and $X = 0$: the last 192 elements of S are random, and the 64 first ones set to 0, then $\Pr[\beta_0 = \beta_1 = 0] \approx 0.06$. Note that, if N is as small as 2, $\Pr[\beta_0 \equiv 2X] \approx 1/256 \approx 0.004$, much higher than the 2^{-32} of an ideal generator. Now consider slightly different states; for a random state where there exists $N \in \{1, \ldots, 253\}$ distinct $i \in \{2, \ldots, 255\}$ such that $S_i = S_0 = S_1 = X$, not necessarily the firsts,

$$\Pr[\beta_0 = 2X] \geq \frac{N+1}{256} \text{ and } \Pr[\beta_1 = 2X] \geq \frac{N}{256}.$$

There are more than $253 \times 2^{8\,096} \geq 2^{8\,103}$ such states, excluding the ones already captured by the previous states mentionned.

Analogously, for a random state where there exists $N \in \{1, \ldots, 254\}$ distinct $i \in \{1, \ldots, 255\}$ such that $S_i = S_0 = X$,

$$\Pr[\beta_0 = 2X] \geq \frac{N}{256}.$$

There are more than $254 \times 2^{8\,128} \geq 2^{8\,135}$ such states. We distinguish this kind of states from the previous one, because the latter can be used by a distinguisher, while the former are much more numerous.

2.2 A strong distinguisher

Based on the weak states presented, a *strong distinguisher* (see Chapter 3 of [Gol01]) is constructed. Briefly, a strong distinguisher is a probabilistic polynomially bounded algorithm, querying two black boxes, each returning a bit sample of fixed length; for one box this sample is truly random, while the

other's is produced by a pseudo-random generator with a random (unknown) initial state.

Here the boxes shall output samples of 64 bits at each query, and the algorithm shall select as the "ISAAC box" the one where the first 32 bits are the most frequently equal to the last 32's (that is, when $\beta_0 = \beta_1$ in ISAAC), and a random box if there is equality of occurences. A random state is weak with probability greater than $2^{8\,192-8\,103} = 2^{-89}$. Thus for a random state,

$$\Pr[\beta_0 = \beta_1 = 2X] \geq 2^{-32} + 2^{-89}\frac{2}{256^2} = 2^{-32} + 2^{-104},$$

whereas this probability is 2^{-32} for a truly random bitstream.

Theorem 4 ([MS01]). *Let D_1, D_2 be distributions, and suppose that the event E happens in D_1 with probability p and un D_2 with probability $p(1+q)$. Then for small p and q, $\mathcal{O}(\frac{1}{pq^2})$ samples suffice to distinguish D_1 from D_2 with constant probability of success.*

Applying this theorem to our distinguisher, we get $p = 2^{-32}$ and $p(1 + q) = 2^{-32} + 2^{-104}$, that is, $q = 2^{-72}$. Hence the distinguisher requires about 2^{176} samples.

2.3 Consequences

For more than $2^{8\,135}$ states, the distribution of the β_i's obtained is far from the uniform one: $2X$ appears with probability greater than 2^{-8}, much higher than the 2^{-32} expected. If such a state is used, one can recover X with probability greater than 1/512, since β_0 takes the value $2X$ with probability greater than 1/256, and there exists two distinct solutions to the equation $2x \equiv 2X$, with unknown x. Moreover, for the first kind of weak states, if N is greater than, say, 216, then $2X$ appears in average more than 90 times, thus X is recovered with high probability, and the random elements can be computed by exhaustive search, so as to find the full state, in 2^{40} iterations of a try-and-check algorithm (there are roughly 2^{72} such states).

3 States with a constant value

When $S_i = X$ for all $i \in \{0, \ldots, 255\}$, and a fixed positive $X < 2^{32}$, as a particular case of the states in Section 6.3.2, we get

$$\Pr[\beta_i \equiv 2X] \geq \frac{256 - i - 1}{256} = 1 - \frac{i+1}{256}.$$

The expected number of i such that $\beta_i \equiv 2X$ is so greater than

$$\sum_{i=0}^{255}(1 - \frac{i+1}{256}) = 127.5.$$

Hence more than half of the elements produced at the first round are $\equiv 2X$ in average, when $S_i = X$ for $i = 0, \ldots, 255$. It is thus straightforward to distinguish between a real random bitstream and a one produced by ISAAC initialized with a state with constant value, since the latter shall have about half of the β_i equal to $2X$. The full state can even be fully recovered, in constant time: the equation $x \equiv 2X$ has two solutions, trivially computed. The right solution is the one that produces β at the first round.

4 Modification of the algorithm

We modify Algorithm .1 to fix the weaknesses stressed (*cf.* line 7).

Input: a, b, c, and the internal state I, an array of 256 32-bit words
Output: an array O of 256 32-bit words

1: $c \to c + 1$
2: $b \to b + c$
3: **for** $i = 0, \ldots, 255$ **do**
4: $\quad x \to S_i$
5: $\quad a \to f(a, i) + I_{(i+128) \mod 256}$
6: $\quad I_i \to a + b + I_{(x \gg 2) \mod 256}$
7: $\quad O_i \to x + a \oplus S_{(I_i \gg 10) \mod 256}$
8: $\quad b \to O_i$
9: **end for**
10: **return** O

Algorithm .3: Modified ISAAC algorithm for an arbitrary round.

This new algorithm has the following properties:

- The three theorems states in Section 6.3.2 do not hold: we get $\beta_0 = S_0 + S_i \oplus (a \ll 13 + S_{128})$, for a random state, S_{128} is random in $\{0, \ldots, 2^{32} - 1\}$, thus so is $a \ll 13 + S_{128}$. This contradicts the two first theorems. The third is trivially contradicted.
- The weak states presented in Section 6.3.2 have $\Pr[\beta_0 \equiv 2X] \approx 2^{-32}$, for the same reasons than previously.
- The probability stated for the states with a constant value does not hold anymore, but the states are still weak: for example, the all-zero state gives $\beta_0 = a \ll 13$ with probability 255/256.

The Diehard battery of tests [Mar95b] is a set of statistical tests for DBRG's, and a success to them is a notorious requirement for a good DBRG. We applyed those tests to 10 samples of 10 Mb of the original and of the modified algorithm, they all successfully passed all the tests. It does not prove nothing, but guarantees a minimal quality of the pseudo-random bitstream.

5 Conclusion

A random state is weak with probability 2^{-57}, which may not be negligible, depending on the application considered. Indeed, weak states might distort simulations, and harm cryptographic applications. In particular, the all-zero state should be avoided. We managed to fix some of the problems pointed out, however the new algorithm does not seem secure either. We hope that these results will help to fill the lack of study of ISAAC, and will inspire deeper analysis.

B The Blum-Goldwasser asymmetric stream cipher

This scheme was designed in 1984 [BG85]: encryption is non-deterministic, and the scheme is IND-CPA secure, assuming the hardness of predicting a sequence of the BBS [BBS86] generator, and of the factorization of a Blum integer (at least as hard as a RSA modulus).

The public-key is a Blum integer $N = pq$ (both p and q must be congruent to 3 modulo 4), and the secret key is the couple factors (p, q). Encryption consists in the generation of a keystream $(b_0, \ldots, b_{\ell-1})$, computed as follows:

1. $r \xleftarrow{\$} \{2, \ell - 1\}$
2. $x_0 \leftarrow r^2 \mod N$
3. For $i = 0, \ldots, \ell - 1$
   (a) $b_i \leftarrow$ least significant bit of x_i
   (b) $x_{i+1} \leftarrow x_i^2 \mod N$
4. $y \leftarrow x_0^{2^\ell} \mod N$

The value y is outputed along with the encrypted message. Given y, p and q, one retreive the pseudo-random sequence $(b_0, \ldots, b_{\ell-1})$ by computing x_0 the following way:

1. $r_p \leftarrow y^{(\frac{p+1}{4})^\ell} \mod p$.
2. $r_q \leftarrow y^{(\frac{q+1}{4})^\ell} \mod q$.
3. $x_0 \leftarrow q(q^{-1} \mod p)r_p + p(p^{-1} \mod q)r_q \mod N$.

Why decryption works ? By Fermat's theorem,

$$x_{i+1}^{\frac{p+1}{4}} = x_i^{\frac{p-1}{2}+1} \equiv x_i \mod p,$$

and so $y = x_0^{2^\ell} \equiv x_\ell \mod N$, which implies $y^{(\frac{p+1}{4})^\ell} \equiv x_0 \mod p$. The Bézout identity gives $q(q^{-1} \mod p) + p(p^{-1} \mod q) \equiv 1 \mod N$, hence,

$$q(x_0 q^{-1} \mod p) + p(x_0 p^{-1} \mod q) \equiv x_0 \mod N.$$

Note that the random value r can also be recovered, by setting the exponents of r_p and r_q to $(\frac{p+1}{4})^{\ell+1}$. Thus it could be considered as a part of the message in a deterministic scheme. If we only consider the pseudo-random stream produced by a secret state r, it can be compared to TCHO2, where the seed is a codeword and the pseudo-random generation is "randomized" by $\mathcal{S}_{\mathcal{L}_P}$ and $\mathcal{S}_\gamma$. The Blum-Goldwasser scheme can also be viewed as a KEM/DEM scheme, where the encapsulated key is r (or x_0), hidden in y,

and the symmetric cipher is a simple XOR with the message. Thus it is not essentially an asymmetric stream cipher, since the first secret recovered thanks to the private key is not the plaintext, but the seed of the BBS generator; its is a trapdoor pseudo-random generator, where the trap allows to recover the seed, not to cancel directly the bitstream as TCHO2 does.

C Number of irreducible and primitive polynomials

To get a sharper expression of the average number of trials before finding a primitive polynomial P in the key generation stage, and for curiosity, we give here some results on the number of irreducible and primitive polynomials on a finite field, mainly taken from the COS (`http://www.theory.cs.uvic.ca/`).

Proposition 13. *The number of irreducible polynomials of degree n over $\mathbb{F}_q$ is*

$$L_q(n) = \frac{1}{n} \sum_{d|n} \mu\left(\frac{n}{d}\right) q^d$$

where μ is the Mőbius function: $\mu(m)$ is equal to 0 if m is not square-free, otherwise $(-1)^k$ with k the number of distinct primes in the factorization of m.

This result is linked with the domain of combinatorics: $L_q(n)$ is also equal to the number of Lyndon words (words that are smaller than any of their right factors, for a lexicographic ordering) of length n on an alphabet of q distinct symbols.

Proposition 14. *The number of primitive polynomials of degree n over $\mathbb{F}_q$ is*

$$P_q(n) = \frac{\phi(q^n - 1)}{n}$$

where ϕ is Euler's totient function.

Thus the probability that a random irreducible binary polynomial of degree n is primitive is

$$\frac{\phi(2^n - 1)}{\sum_{d|n} \mu\left(\frac{n}{d}\right) 2^d},$$

note that, if $n = \prod_{p_i \in \mathbb{P}} p_i^{\alpha_i}$, there are $\prod_i (\alpha_i + 1)$ divisors of n.

Example 12. *Then there are exactly* 52 377 *irreducible and* 24 000 *primitive binary polynomials of degree* 20 *(*45 %*), and respectively* 99 858 *and* 84 672 *of degree* 21 *(*89 %*). For prime degrees leading to a Mersenne prime, there as as many irreducible as primitive polynomials.*

A known result states that $L_2(n)$ can be asymptotically approximated to $2^n/n$. We verify this experimentaly: for degrees in $[1, 200]$ the average error fraction of the real value is roughly 0.015, whereas it is about $8.65 \cdot 10^{-17}$ for degrees in $[800, 1\,000]$.

Proposition 15 ([GM01]). *The exact number of multiples of weight v (with constant term 1) of any* primitive *polynomial of degree t is*

$$N_{d,v} = \frac{1}{v-1}\binom{2^t-2}{v-2} - N_{t,v-1} - \frac{v-1}{v-2}(2^t - v + 1)N_{t,v-2}$$

with initial conditions $N_{t,1} = N_{t,2} = 0$.