Linear Bandwidth Naccache-Stern Encryption

Benoît Chevallier-Mames
DCSSI,
51, Boulevard de la Tour Maubourg
75700 Paris, France
benoit.chevallier-mames@sgdn.gouv.fr

David Naccache, Jacques Stern
École normale supérieure
45 rue d'Ulm
F-75230 Paris CEDEX 05, France
{david.naccache,jacques.stern}@ens.fr

Abstract. The Naccache-Stern (NS) knapsack cryptosystem is a public-key encryption scheme which is not widely known or used, despite (or because of) its original design. In this scheme, the ciphertext is obtained by multiplying the public-keys indexed by the message bits modulo a system parameter p. The cleartext is then recovered by factoring the ciphertext raised to a secret power modulo p.
NS encryption requires one multiplication per two bits on the average, while decryption is roughly as costly as an RSA decryption. However, NS features a bandwidth sublinear in $\log p$, namely $\log p / \log\log p$.
This paper presents new NS variants allowing to reach bandwidths *linear* in $\log p$. The price to pay for reaching a linear bandwidth is a public-key of size $\log^3 p / \log\log p$. We hope that these modifications will make the NS knapsack cryptosystem more practical and attracting, allowing its use in more cryptographic protocols.

1 Introduction

The Naccache-Stern cryptosystem (NS), introduced a decade ago in [NS97], is a public-key cryptosystem based on the following problem:

$$\text{given } p,\ c \text{ and a set } \{p_i\},\ \text{find a binary vector } x \text{ such that } c = \prod_{i=0}^{n-1} p_i^{x_i} \bmod p.$$

Trivially, if the p_i-s are relatively prime and much smaller than p, the above problem can be solved in polynomial time by factoring c in $\mathbb{N}$.

A trapdoor is obtained by extracting a secret (s-th) modular root of each p_i and publishing these roots, denoted $v_i = \sqrt[s]{p_i} \bmod p$. By raising a product of such roots to the s-th power, each v_i shrinks back to a much smaller p_i and x can be found by factoring the result in $\mathbb{N}$.

Unfortunately, no security proofs linking NS's security to standard complexity assumptions are known, but at the same time, no efficient chosen-plaintext attacks against NS's one-wayness are known either.

More formally, let p be a large public prime[1] and denote by n the largest integer such that:

$$p > \prod_{i=0}^{n-1} p_i \text{ where } p_i \text{ is the } i\text{-th prime (start from } p_0 = 2).$$

[1] For technical reasons, p must be a safe prime, *cf.* to Section 2.4 of [NS97] or Section 5.

The secret-key $0 < s < (p-1)$ is a random integer such that $\gcd(p-1, s) = 1$ and the public-keys are the n roots:

$$v_i = \sqrt[s]{p_i} \bmod p.$$

A message $m = \sum_{i=0}^{n-1} 2^i \, m_i$, where $m_i \in \{0, 1\}$, is encrypted as $c = \prod_{i=0}^{n-1} v_i^{m_i} \bmod p$ and recovered by:

$$m = \sum_{i=0}^{n-1} \frac{2^i}{p_i - 1} \times \Big(\gcd(p_i, c^s \bmod p) - 1 \Big).$$

Denoting by $\ln(x)$ natural logarithms and by $\log(x)$ base-2 logarithms, it is easy to see that NS's bandwidth is sublinear: As $p_i \sim i \ln i$, we have

$$\ln p \sim \sum_{i=0}^{n} \ln p_i \sim n \ln n \;\; \Rightarrow \;\; \ln \ln p \sim \ln n,$$

which in turn gives:

$$n \sim \frac{\ln p}{\ln \ln p} \sim \frac{\log p}{\log \log p}.$$

In a typical setting, a 1024-bit p corresponds to a sixteen kilobyte public-key and allows encrypting 131-bit messages.

[NS97] also describes a variant depending on a parameter $\ell \in \mathbb{N}$. Here, p is such that

$$p > \prod_{i=0}^{n-1} {p_i}^{\ell}.$$

$m = \sum_{i=0}^{n-1} (\ell+1)^i \, m_i$, expressed in base $(\ell+1)$ (here $m_i \in [0, \ell]$), is encrypted as

$$c = \prod_{i=0}^{n-1} v_i^{m_i} \bmod p,$$

and decryption is straightforwardly modified. In this paper, we refer this version as the "$(\ell+1)$-base variant".

The goal of this work is to improve the scheme's bandwidth using more sophisticated arithmetic encoding techniques. In next section, we propose a technique based on modular fractions that multiplies bandwidth by $\log_2 3 \simeq 1.58$ for binary-message NS.[2] Section 3 describes a new message encoding technique that dramatically increases bandwidth (to become linear in $\log p$). Section 4 extends the previous idea to $(\ell+1)$-base NS, thereby further increasing bandwidth. In Section 5, we have a look on the security issues of the proposed improvements. Finally, Section 6 describes a number of combinatorial problems whose solutions would yield even more efficient NS variants.

[2] The factor becomes $\log_{1+\ell}(2\ell+1)$ for $(\ell+1)$-base variant.

2 Fractional Message Encoding

In this section, we show how to add signs to message bits for free. Consider a message represented in a signed binary notation, *i.e.*,

$$m = \sum_{i=0}^{n-1} 2^i \, m_i \quad \text{where } m_i \in \{-1, 0, 1\}$$

and an unchanged encryption procedure. During decryption, the receiver will find a u such that:

$$u = c^s = \frac{a}{b} \mod p, \qquad \text{with } \begin{cases} a = \prod_{m_i=1} p_i \\ b = \prod_{m_i=-1} p_i. \end{cases}$$

The following theorem shows that, given u, one can recover a and b efficiently using Gauss's algorithm for finding the shortest vector in a two-dimensional lattice [Val91].

Theorem 1 ([FSW02]). *Let $a, b \in \mathbb{Z}$ such that $|a| \leqslant A$ and $0 < b \leqslant B$. Let p be a prime such that $2AB < p$. Let $u = a/b \mod p$. Then given $\{A, B, u, p\}$, one can recover $\{a, b\}$ in polynomial time.*

Taking $A = B = \lfloor \sqrt{p} \rfloor - 1$, we have that $2AB < p$. If we assume in addition that m was such that $0 \leqslant a \leqslant A$ and $0 < b \leqslant B$, we can recover a and b from s in polynomial time. And by testing the divisibility of a and b by the small primes p_i, the receiver can eventually recover m as before.

But what happens if $|a| > A$ or $b > B$?

To solve this case too, let us tweak the definition of p to $p > 2^w \times \prod_{i=0}^{n-1} p_i$, for some small integer $w \geqslant 1$ (we suggest to take $w = 50$), and define a finite sequence $\{A_i, B_i\}$ of integers such that:

$$A_i = 2^{wi} \quad \text{and} \quad B_i = \left\lfloor \frac{p-1}{2A_i} \right\rfloor.$$

For all $i > 0$, we have that $ab < 2A_iB_i < p$. Moreover, there must exist at least one index i such that $0 \leqslant a \leqslant A_i$ and $0 < b \leqslant B_i$. Then using the algorithm of Theorem 1, given A_i, B_i, p and s, one can recover a and b, and eventually recover m. The problem is that we have just lost the guarantee that such an $\{a, b\}$ is unique. Namely, we could in theory hit another $\{a', b'\}$ whose ratio gives the same u, for some other index $i' \neq i$. But we expect this to happen with negligible probability for large enough w.

Senders wishing to eliminate even the negligible probability that decryption will produce more than one plaintext can still simulate the decryption's Gaussian phase and, in case, re-randomize m until decryption is unambiguous.

The effect of this optimization is noteworthy as for the price of a constant increase (*e.g.* $\simeq 50$ bits) in p, bandwidth is multiplied by a factor of $\log_2 3$.

Note that as $\ell > 1$ is used in conjunction with this technique (*i.e.,* in the $(\ell + 1)$-base variant), bandwidth improvement tends to 1 bit per prime as ℓ grows. Namely, fractional encoding increases bandwidth from $n \log(1 + \ell)$ to $n \log(1 + 2\ell)$.

3 Small Prime Packing

3.1 Description

Let the integer $\gamma \geqslant 2$ be a system parameter. We now group the small primes p_i into n packs containing γ small primes each.[3] That is, the first pack will contain primes p_1 to p_γ, the second pack will include primes $p_{\gamma+1}$ to $p_{2\gamma}$ *etc.* As previously, the p_i-s are indexed in increasing order.

We also update the condition on the large prime p to:

$$\prod_{i=1}^{n} p_{\gamma i} < p.$$

In other words, we do not request anymore p to be larger than the product of all the small primes. Instead, we *only* request p to be larger than the product of the largest representatives of each pack.

We now represent m in base γ, *i.e.,*

$$m = \sum_{i=0}^{n-1} \gamma^i \, m_i \quad \text{where} \quad m_i \in [0, \gamma - 1]$$

and encode m by picking in pack i the prime representing the message's i-th digit m_i and multiplying all so chosen p_i-s modulo p:

$$\mathsf{encoding}(m) = \prod_{i=0}^{n-1} p_{\gamma i + m_i + 1} \bmod p.$$

We can now apply this encoding to the NS and re-define encryption as:

$$c = \mathsf{encryption}(m) = \prod_{i=0}^{n-1} v_{\gamma i + m_i + 1} \bmod p.$$

To decrypt c, the receiver computes $u = c^s \bmod p$ and recovers m by factoring u. Note that as soon as a representative of pack i is found, the receiver can stop sieving within pack i and start decrypting digit $i + 1$.

3.2 A Small Example

We illustrate the mechanism by a small toy-example.

[3] For simplicity of reading, we now set the first prime as $p_1 = 2$.

• KEY GENERATION FOR $n = 3$ AND $\gamma = 4$:

The prime $p = 4931 > p_\gamma \times p_{2\gamma} \times p_{3\gamma} = 7 \times 19 \times 37$ and the secret $s = 3079$ yield the v-list:

$$\text{pack 1}\begin{cases} v_1 = \sqrt[s]{2} \bmod p = 1370 \\ v_2 = \sqrt[s]{3} \bmod p = 1204 \\ v_3 = \sqrt[s]{5} \bmod p = 1455 \\ v_4 = \sqrt[s]{7} \bmod p = 3234 \end{cases} \quad \text{pack 2}\begin{cases} v_5 = \sqrt[s]{11} \bmod p = 2544 \\ v_6 = \sqrt[s]{13} \bmod p = 3366 \\ v_7 = \sqrt[s]{17} \bmod p = 1994 \\ v_8 = \sqrt[s]{19} \bmod p = 3327 \end{cases} \quad \text{pack 3}\begin{cases} v_9 = \sqrt[s]{23} \bmod p = 4376 \\ v_{10} = \sqrt[s]{29} \bmod p = 1921 \\ v_{11} = \sqrt[s]{31} \bmod p = 3537 \\ v_{12} = \sqrt[s]{37} \bmod p = 3747 \end{cases}$$

• ENCRYPTION OF $m = 50$:

We start by writing m is base $\gamma = 4$, *i.e.*, $m = 50 = 302_4$ and encrypt it as:

$$c = v_{(0\cdot 4+3+1)} \times v_{(1\cdot 4+0+1)} \times v_{(2\cdot 4+2+1)} = v_4 \times v_5 \times v_{11} \bmod 4931 = 4484.$$

• DECRYPTION:

By exponentiation, the receiver retrieves:

$$c^s \bmod p = 4484^{3079} \bmod 4931 = 7{\times}11{\times}31 = p_{(0\cdot 4+3+1)} {\times} p_{(1\cdot 4+0+1)} {\times} p_{(2\cdot 4+2+1)},$$

whereby $m = 302_4$.

3.3 Bandwidth Considerations

The bandwidth gain stems from the fact that, for large i, we have $p_{\gamma i+1} \simeq p_{\gamma i+\gamma}$ which allows the new format to accommodate $\log_2 \gamma$ message bits at the price of one single $p_{\gamma i+\gamma}$. This situation is much more favorable than the original NS, where each message bit costs a new p_i.

More precisely, $p_{\gamma i} \sim \gamma i \ln i$ yields an $(n \log \gamma)$-bit bandwidth where:

$$n \sim \ln p / \ln \ln p \sim \log p / \log \log p$$

The bandwidth gain is thus a constant multiplicative factor (namely $\log \gamma$) and the increase in n is logarithmic. Note that at the same time, the v_i-list becomes γ times longer.

The following table shows the performances of the new encoding rule for a 1024-bit p. The first row represents the features of the original NS for the sake of comparison.

γ	n	plaintext bits $= n \log \gamma$	public key size $= \gamma n \log p$	information rate$= n\frac{\log \gamma}{\log p}$
NS	131	131 bits	16 kilobytes	0.13%
2	116	116 bits	29 kilobytes	0.11%
4	104	208 bits	52 kilobytes	0.20%
8	94	282 bits	94 kilobytes	0.28%
16	86	344 bits	172 kilobytes	0.34%
32	79	395 bits	316 kilobytes	0.39%
64	73	438 bits	584 kilobytes	0.43%
128	68	476 bits	1088 kilobytes	0.46%
256	64	512 bits	2048 kilobytes	0.50%
512	60	540 bits	3840 kilobytes	0.53%
1024	56	560 bits	7168 kilobytes	0.55%

As one can see, the bandwidth improvement is significant.

3.4 Linear Bandwidth

Setting $\gamma = n$ (*i.e.*, n packs containing n primes each), we can approximate:

$$p_{\gamma i} \sim \gamma i \ln i \sim ni \ln i \;\Rightarrow\; \sum_{i=0}^{n} \ln p_{\gamma i} \sim \sum_{i=0}^{n} \ln(n^2 \ln n) \sim n \ln(n^2) \sim 2n \ln n \sim \ln p.$$

As $\ln n \sim \ln \ln p$, we get an $n \log n$ bit bandwidth with:

$$n \sim \frac{\ln p}{2 \ln \ln p} \sim \frac{\log p}{2 \log \log p}.$$

Substituting the expressions of n and $\log n$ into the bandwidth formula (that is $n \log n$), we see that the resulting information rate turns out to be $\frac{1}{2}$. This encoding scheme therefore features a linear bandwidth, while NS is only sublinear. Note that this format is compatible with fractional encoding (Section 2), thereby allowing further bandwidth gains.

3.5 Optimizing the Encoding of Zeros

We now observe that the encoding of zeros does not require using new primes. The corresponding tweak to the encryption procedure is straightforward and allows to lower the number of p_i-s from γn to $(\gamma - 1)n$. This increases n and hence the information rate.

For the previous toy-example, the packs will become:

$$\text{pack 1} \begin{cases} v_1 = 1 \\ v_2 = \sqrt[s]{2} \bmod p \\ v_3 = \sqrt[s]{3} \bmod p \\ v_4 = \sqrt[s]{5} \bmod p \end{cases} \quad \text{pack 2} \begin{cases} v_5 = 1 \\ v_6 = \sqrt[s]{7} \bmod p \\ v_7 = \sqrt[s]{11} \bmod p \\ v_8 = \sqrt[s]{13} \bmod p \end{cases} \quad \text{pack 3} \begin{cases} v_9 = 1 \\ v_{10} = \sqrt[s]{17} \bmod p \\ v_{11} = \sqrt[s]{19} \bmod p \\ v_{12} = \sqrt[s]{23} \bmod p \end{cases}$$

p can now be chosen as $p = 1499 > p_\gamma \times p_{2\gamma} \times p_{3\gamma} = 5 \times 13 \times 23$, which is indeed somewhat shorter than the modulus used in Section 3.2.

Figures are given in the following table, where the first row (*i.e.*, $\gamma = 2$) represents the original NS. As before, this results stand for a 1024-bit p. The optimization is particularly interesting for small γ values.

γ	n	plaintext bits $= n \log \gamma$	public key size $= (\gamma - 1)\, n \log p$	information rate$= n \frac{\log \gamma}{\log p}$
2	131	131 bits	16 kilobytes	0.13% (original NS)
4	108	216 bits	40 kilobytes	0.21%
8	96	288 bits	84 kilobytes	0.28%
16	86	344 bits	161 kilobytes	0.34%
32	79	395 bits	306 kilobytes	0.39%
64	73	438 bits	575 kilobytes	0.43%
128	68	476 bits	1080 kilobytes	0.46%
256	64	512 bits	2040 kilobytes	0.50%
512	60	540 bits	3832 kilobytes	0.53%
1024	57	570 bits	7289 kilobytes	0.56%

4 Using Powers of Primes

In this section we apply prime-packing to the $(\ell+1)$-base variant. We start with an example, to explain as simply as possible the obtained scheme.

4.1 A Small Example

Take $n = 1$ and $\gamma = 4$, *i.e.* a single pack, containing $\{p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7\}$. We also set $\ell = 2$, pick a modulus $p > 7^\ell = 7^2 = 49$, define the public key as:

$$\{v_1 = \sqrt[s]{2} \bmod p, v_2 = \sqrt[s]{3} \bmod p, v_3 = \sqrt[s]{5} \bmod p, v_4 = \sqrt[s]{7} \bmod p\}$$

and consider all p_i products of weight smaller or equal to ℓ:

$$\begin{array}{lll}
7^0 \times 5^0 \times 3^0 \times 2^0 & 7^0 \times 5^0 \times 3^0 \times 2^1 & 7^0 \times 5^0 \times 3^0 \times 2^2 \\
7^0 \times 5^0 \times 3^1 \times 2^0 & 7^0 \times 5^0 \times 3^1 \times 2^1 & 7^0 \times 5^0 \times 3^2 \times 2^0 \\
7^0 \times 5^1 \times 3^0 \times 2^0 & 7^0 \times 5^1 \times 3^0 \times 2^1 & 7^0 \times 5^1 \times 3^1 \times 2^0 \\
7^0 \times 5^2 \times 3^0 \times 2^0 & 7^1 \times 5^0 \times 3^0 \times 2^0 & 7^1 \times 5^0 \times 3^0 \times 2^1 \\
7^1 \times 5^0 \times 3^1 \times 2^0 & 7^1 \times 5^1 \times 3^0 \times 2^0 & 7^2 \times 5^0 \times 3^0 \times 2^0
\end{array}$$

All in all, we have $1 + 4 + 10 = \binom{\gamma+\ell}{\ell} = 15$ products[4] that can be associated to 15 message digit values. Therefore, to encode a message digit $m_0 \in [0, 14]$, we use any unranking algorithm [SW86] returning $\mathsf{unrank}(m_0) = \{a, b, c, d\}$ and encrypt m_0 as:

$$c = \mathsf{encryption}(m_0) = {v_1}^a \times {v_2}^b \times {v_3}^c \times {v_4}^d \bmod p.$$

For instance, using a lexicographic ranking of words of weight two:

$$\begin{array}{lcl}
\mathsf{unrank}(\ 0) = \{0,0,0,0\} & \rightsquigarrow & 7^0 \times 5^0 \times 3^0 \times 2^0 \\
\mathsf{unrank}(\ 1) = \{0,0,0,1\} & \rightsquigarrow & 7^0 \times 5^0 \times 3^0 \times 2^1 \\
\mathsf{unrank}(\ 2) = \{0,0,0,2\} & \rightsquigarrow & 7^0 \times 5^0 \times 3^0 \times 2^2 \\
\mathsf{unrank}(\ 3) = \{0,0,1,0\} & \rightsquigarrow & 7^0 \times 5^0 \times 3^1 \times 2^0 \\
\mathsf{unrank}(\ 4) = \{0,0,1,1\} & \rightsquigarrow & 7^0 \times 5^0 \times 3^1 \times 2^1 \\
\mathsf{unrank}(\ 5) = \{0,0,2,0\} & \rightsquigarrow & 7^0 \times 5^0 \times 3^2 \times 2^0 \\
\mathsf{unrank}(\ 6) = \{0,1,0,0\} & \rightsquigarrow & 7^0 \times 5^1 \times 3^0 \times 2^0 \\
\mathsf{unrank}(\ 7) = \{0,1,0,1\} & \rightsquigarrow & 7^0 \times 5^1 \times 3^0 \times 2^1 \\
\mathsf{unrank}(\ 8) = \{0,1,1,0\} & \rightsquigarrow & 7^0 \times 5^1 \times 3^1 \times 2^0 \\
\mathsf{unrank}(\ 9) = \{0,2,0,0\} & \rightsquigarrow & 7^0 \times 5^2 \times 3^0 \times 2^0 \\
\mathsf{unrank}(10) = \{1,0,0,0\} & \rightsquigarrow & 7^1 \times 5^0 \times 3^0 \times 2^0 \\
\mathsf{unrank}(11) = \{1,0,0,1\} & \rightsquigarrow & 7^1 \times 5^0 \times 3^0 \times 2^1 \\
\mathsf{unrank}(12) = \{1,0,1,0\} & \rightsquigarrow & 7^1 \times 5^0 \times 3^1 \times 2^0 \\
\mathsf{unrank}(13) = \{1,1,0,0\} & \rightsquigarrow & 7^1 \times 5^1 \times 3^0 \times 2^0 \\
\mathsf{unrank}(14) = \{2,0,0,0\} & \rightsquigarrow & 7^2 \times 5^0 \times 3^0 \times 2^0
\end{array}$$

[4] The attentive reader would rightly note that there are actually more p_i products smaller than p. This is true for very small primes in the first packs, but when one considers packs where the minimal and maximal p_i-s are roughly equivalent in size, the number of products quickly tends to $\binom{\gamma+\ell}{\ell}$.

$m_0 = 12$ will be encrypted as $\mathsf{encryption}(12) = \sqrt[s]{3} \times \sqrt[s]{7} = v_2 \times v_4 \bmod p$. Decryption recovers $2^0 \times 3^1 \times 5^0 \times 7^1$ by exponentiation and determines that $m_0 = \mathsf{rank}(\{1, 0, 1, 0\}) = 12$.

The improvement of the method comes from the fact that we encrypt $\log(15)$ bits where the $(\ell + 1)$-base variant only encrypts $\log(3)$. In other words, the prime-packing idea fits particularly well to the $(\ell + 1)$-base system.

Also, as is all practical instances $\binom{\gamma+\ell}{\ell}$ will remain moderate (typically less than one hundred), functions $\mathsf{rank}(\cdot)$ and $\mathsf{unrank}(\cdot)$ can be implemented as simple lookup tables rather than full-fledged constructive combinatorial algorithms.

4.2 Formal Description

Let us describe now the scheme formally. Let $\ell \geqslant 1$ and γ be two integer parameters[5] and consider n packs containing γ small primes each (the primes starting from $p_1 = 2$). We pick a prime p such that:

$$\prod_{i=1}^{n} {p_{\gamma i}}^{\ell} < p.$$

A classical result[6] shows that there are $\binom{\gamma+\ell}{\ell}$ different γ-tuples $\{d_1, \ldots, d_\gamma\}$ such that $0 \leqslant d_k$ and $\sum_k d_k \leqslant \ell$. We thus define $\mathsf{unrank}(\cdot)$ as an invertible function mapping integers in $[0, \binom{\gamma+\ell}{\ell} - 1]$ to $\{d_1, \ldots, d_\gamma\}$-tuples.

To encrypt a message expressed in base $\binom{\gamma+\ell}{\ell}$, *i.e.*, $m = \sum_{i=0}^{n-1} \binom{\gamma+\ell}{\ell}^i m_i$ with $m_i \in [0, \binom{\gamma+\ell}{\ell} - 1]$ one computes:

$$c = \mathsf{encryption}(m) = \prod_{i=0}^{n-1} \prod_{j=1}^{\gamma} {v_{\gamma i+j}}^{d_{i,j}} \bmod p \quad \text{where} \quad \{d_{i,1}, \ldots, d_{i,\gamma}\} = \mathsf{unrank}(m_i)$$

To decrypt the ciphertext factor $c^s \bmod p$ in $\mathbb{N}$ and recover each m_i by:

$$m_i = \mathsf{rank}(\{d_{i,1}, \ldots, d_{i,\gamma}\})$$

4.3 Bandwidth Considerations

The table below shows that the variant described in this Section features a better bandwidth and smaller public-keys than the basic prime-packs encoding of Section 3. Data was generated for several public-key sizes (namely 10, 20, 50, and 500 kilobytes) and a 1024-bit p, being reminded that first line $(\gamma, \ell) = (1, 1)$ is the original NS:

[5] NS corresponds to the case $\{\gamma, \ell\} = \{1, 1\}$ and the $(\ell + 1)$-base variant corresponds to $\gamma = 1$.

[6] Proof is given in Appendix A.

γ	ℓ	n	plaintext bits $= n \log \binom{\gamma+\ell}{\ell}$	public key size $= \gamma n \log p$	information rate$= n \frac{\log \binom{\gamma+\ell}{\ell}}{\log p}$
1	1	131	131 bits	16 kilobytes	0.13%
3	9	17	132 bits	7 kilobytes	0.13%
5	10	14	162 bits	10 kilobytes	0.16%
10	8	15	231 bits	20 kilobytes	0.23%
12	10	12	232 bits	20 kilobytes	0.23%
39	10	10	329 bits	50 kilobytes	0.32%
570	10	7	489 bits	500 kilobytes	0.48%

5 About the Influence of Our Improvements on Security

As we already said, the original NS did not come with a proof of security, and we have no hope nor claim that our modifications may correct this lack. In this section however, we recapitulate certain facts or arguments of security, some of which are already known since [NS97], for the clarity and the self-containment of this paper.

5.1 What Security Can Be Attained?

The most important security notion that one would expect from an encryption scheme to fulfil is the property of *one-wayness* (OW): an attacker should not be able to recover the plaintext matching a given ciphertext. We capture this notion more formally by saying that for any adversary $\mathcal{A}$, succeeding in inverting the effects of encryption on a ciphertext c should occur with negligible probability.

The notion of *semantic security* (IND) [GM84], as known as *indistinguishability of encryptions* captures a stronger notion of privacy. The adversary $\mathcal{A}$ is said to break IND when, after choosing two messages m_0 and m_1 of same length, he can decide whether a given ciphertext corresponds to m_0 or m_1. An encryption scheme is said to be semantically secure or indistinguishable if no probabilistic algorithm can break IND.

The NS cryptosystem, or the variants we have presented in Section 2, 3 or 4 can not ensure indistinguishability, since they are by nature deterministic. The hope however is that there may be one-way. To achieve full-security with ours variants (or with NS), one would use generic transformations as those by Fujisaki and Okamoto in [FO99,FO00]: nevertheless, as there is no formal reduction from a known recognized cryptographic problem to an attack of a NS-type scheme (either in the original form or in the variants we propose), the application of these generic rules will not achieve a provably-secure scheme, but give only empirical arguments of security.

5.2 Security Arguments of Our Variants and Discussions

OUR SCHEMES CAN BE BROKEN IF ONE SOLVES THE DISCRETE-LOGARITHM. It is clear that an attacker that has an access to a discrete-logarithm oracle can easily and totally break the NS scheme or the variants presented in this paper.

Indeed, to this aim, it is sufficient to ask the oracle for the discrete-logarithm of p_1 in base v_1, which actually is the secret key s of the scheme. Even if the primes are hidden or sorted in a non-conventional order to obscure the attacker, the fact that primes must be small for efficiency makes that they are easily guessable, and the attacker can get the secret key after few tries.

Larger message space may make NS-type problems harder. As one can see, the schemes presented in this paper are — as original NS — multiplicative knapsacks. Even if there is no known efficient algorithm to solve this kind of problem, one has to ensure that a brute-force attack consisting in testing all the products is impossible. More precisely, getting back the arguments of Naccache and Stern in Section 2.3 of [NS97], it is at least required that the message space is larger than 160 bits, if one wants 80-bit security. In this sense, our bandwidth improvements go in the direction of better security: the larger the message space, the stronger the security may be. However, we cannot provably claim that the variants are stronger, as improvements come also with larger public-keys, which — at least in the information theoretic sense — give more information to the attacker about the secret key.

About small factors of $(p-1)$. As depicted in Section 2.4 of [NS97], the small factors of $(p-1)$ are of importance. To rephrase Naccache and Stern, noting $\mathbb{QR}_p$ and $\overline{\mathbb{QR}}_p$ the quadratic and non-quadratic residues modulo p respectively, imagine one has a NS ciphertext

$$c = \prod_{i=0}^{n-1} v_i^{m_i} \bmod p.$$

By computing $a = c^{\frac{p-1}{2}} \bmod p$, one will get

$$a = \prod_{v_i \in \overline{\mathbb{QR}}_p} (-1)^{m_i} \bmod p,$$

which in turn leaks the value $\sum_{v_i \in \overline{\mathbb{QR}}_p} m_i \bmod 2$. This partial recovery of information about message can also be declined with other small factors of $(p-1)$. Therefore, the authors of [NS97] advised to prefer strong prime p, and to use one of the bit of the message space to cancel the leakage (in other words, they simply make that $\sum_{v_i \in \overline{\mathbb{QR}}_p} m_i \bmod 2$ is constant).

For our variants, it is not as simple to use same kind of attacks. Indeed, in a given prime pack, there might be some primes in $\mathbb{QR}_p$ and others in $\overline{\mathbb{QR}}_p$. For example, with the variant of Section 3, getting $c = \mathsf{encryption}(m) = \prod_{i=0}^{n-1} v_{\gamma i + m_i + 1} \bmod p$, the attacker may compute $a = c^{\frac{p-1}{2}} \bmod p$. As

$$a = \prod_{v_{\gamma i + m_i + 1} \in \overline{\mathbb{QR}}_p} (-1) \bmod p,$$

this gives the attacker the parity of the number of message digits m_i so that their corresponding primes is in $\overline{\mathbb{QR}}_p$. Even if the information is less precise than in NS case, we however consider this as a possible leak, and, since there is a light protection, we recommend also to use strong prime for our variants.

ABOUT THE POSSIBILITY OF A REDUCTION FROM ATTACKING NS TO ATTACKING OUR VARIANTS. At first sight, it might be believed that there exists reductions between NS and the variants we proposed: indeed, one may hope that if we have an access to a decryption oracle $\mathcal{D}$ of one of our scheme, one can build a NS decryption oracle $\mathcal{D}'$.

However, a simple observation shows that it is certainly not possible: in the variants we propose, the public key is longer, made of more elements related to the secret key. Therefore, from a NS public key and challenge, it may certainly be possible to build a challenge for our variants, but there is very little hope that one can construct the whole correct public key.

Thus, we have no formal proof or argument that the difficulty of original NS or the variants proposed in this paper is equivalent or related, even if it *looks* — if the additional public keys we give for our improvements do not leak too much about the secret key — both original NS and our variants are similar uses of multiplicative knapsacks, with an advantage for our solutions being that the number of possible products is much larger.

6 Further Research and Open Problems

We conclude this paper with a couple of interesting combinatorial problems whose solution might further improve the NS's bandwidth.

Setting $\ell = 1$, not all collections of γn integers allow encoding γ^n combinations. Let $\mathcal{A} = \{A_1, ..., A_n\}$ be n integer-sets, each of size γ and denote by $A_i[j]$ the j-th element of A_i. We call $\mathcal{A}$ an *encoder* if its A_i-s can be used as a collection of packs encoding exactly $n \log_2 \gamma$ bits, or, in other words, if no collisions in the integer sub-products of $\mathcal{A}$ occur. Improving the NS consists in finding "better" encoders.

To compare encoders, we use their *head-products*, namely:

$$h(\mathcal{A}) = \prod_{i=1}^{n} \max_j (A_i[j])$$

Head-products lower-bound the modulus p and hence "measure" bandwidth.

We saw that when the $A_i[j]$ are the first small primes, $\mathcal{A}$ is an encoder and $h(\mathcal{A}) = \prod_{i=1}^{n} p_{i\gamma}$ (Section 3). We also saw that when the smallest element in each A_i is one, the resulting $\mathcal{A}$ is still an encoder whose head-product is $h(\mathcal{A}) = \prod_{i=1}^{n} p_{i(\gamma-1)}$ (Section 3.5).

This gives raise to interesting combinatorial problems such as finding algorithms for efficiently testing that a given $\mathcal{A}$ is an encoder, or finding algorithms for constructing optimal encoders, *i.e.* encoders featuring a minimal head-product (and consequently a maximal bandwidth).

As an example, a (rather inefficient) computer-aided exploration for $n = 3$ and $\gamma = 4$ discovered the optimal encoder $\mathcal{A}$ whose $h(\mathcal{A}) = 4 \times 8 \times 13 = 416$:

$$\begin{array}{lll} \text{pack 1} \begin{cases} A_1[1] = 1 \\ A_1[2] = 2 \\ A_1[3] = 3 \\ A_1[4] = 4 \end{cases} & \text{pack 2} \begin{cases} A_2[1] = 1 \\ A_2[2] = 5 \\ A_2[3] = 7 \\ A_2[4] = 8 \end{cases} & \text{pack 3} \begin{cases} A_3[1] = \ 1 \\ A_3[2] = \ 9 \\ A_3[3] = 11 \\ A_3[4] = 13 \end{cases} \end{array}$$

Interestingly, this encoder contains primes, but also powers of primes. Moreover, throughout our search, non-optimal encoders containing composite integers (such as 6) were found as well.

Decoding messages encoded with such complicated $\mathcal{A}$-s might not always be straightforward as in such atypical encoders, decoding is based on impossibilities of certain factor combinations rather than on the occurrence of certain factors in the product.

The above questions also generalize to packs of rationales.

References

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

[FO00] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. *IEICE Transaction of Fundamentals of Electronic Communications and Computer Science*, E83-A(1):24–32, 2000.

[FSW02] Pierre-Alain Fouque, Jacques Stern and Jan-Geert Wackers. Cryptocomputing with rationals. In *Financial Cryptography – FC 2002*, volume 2357 of *Lecture Notes in Computer Science*, pages 136–146. Springer-Verlag, 2002.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. In *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[NS97] David Naccache and Jacques Stern. A new public-key cryptosystem. In *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 27–36. Springer-Verlag, 1997.

[SW86] Dennis Stanton and Dennis White. Constructive combinatorics, Springer-Verlag New York, Inc., New York, NY, 1986.

[Val91] Brigitte Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12(4):556–572, 1991.

A Proof of Theorem 1

In this appendix, we remind how we evaluate the number, denoted $\mathcal{R}_{\ell,\gamma}$, of different γ-tuples $\{d_1, \ldots, d_\gamma\}$ such that $0 \leqslant d_k$ and $\sum_k d_k \leqslant \ell$.

The number of sequences of γ integers whose sum equals i is $\binom{\gamma+i-1}{i}$. Therefore, we have:

$$\mathcal{R}_{\ell,\gamma} = \sum_{i=0}^{\ell} \binom{\gamma + i - 1}{i}.$$

Assume that we have $\mathcal{R}_{\ell,\gamma} = \binom{\gamma+\ell}{\ell}$. What happens for $(\ell+1)$?

$$\mathcal{R}_{\ell+1,\gamma} = \sum_{i=0}^{\ell+1} \binom{\gamma+i-1}{i} = \mathcal{R}_{\ell,\gamma} + \binom{\gamma+\ell+1-1}{\ell+1} = \binom{\gamma+\ell}{\ell} + \binom{\gamma+\ell}{\ell+1} = \binom{\gamma+\ell+1}{\ell+1}$$

where the last line stem's from Pascal's rule.

As $\mathcal{R}_{0,\gamma} = 1 = \binom{\gamma}{0}$, we get by induction that:

$$\mathcal{R}_{\ell,\gamma} = \binom{\gamma+\ell}{\ell}.$$