Linear Bandwidth Naccache-Stern Encryption

Benoît Chevallier-Mames[1], David Naccache[2], and Jacques Stern[2]

[1] DCSSI, Laboratoire de cryptographie,
51, Boulevard de la Tour Maubourg,
F-75700 Paris, France
benoit.chevallier-mames@sgdn.gouv.fr

[2] 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 an original yet little-known public-key encryption scheme. In this scheme, the ciphertext is obtained by multiplying public-keys indexed by the message bits modulo a prime p. The cleartext is recovered by factoring the ciphertext raised to a secret power modulo p.
NS encryption requires a multiplication per two plaintext bits on the average. Decryption is roughly as costly as an RSA decryption. However, NS features a bandwidth sublinear in $\log p$, namely $\log p / \log\log p$. As an example, for a 2048-bit prime p, NS encryption features a 233-bit bandwidth for a 59-kilobyte public key size.
This paper presents new NS variants achieving bandwidths *linear* in $\log p$. As linear bandwidth claims a public-key of size $\log^3 p / \log\log p$, we recommend to combine our scheme with other bandwidth optimization techniques presented here.
For a 2048-bit prime p, we obtain figures such as 169-bit plaintext for a 10-kilobyte public key, 255-bit plaintext for a 20-kilobyte public key or a 781-bit plaintext for a 512-kilobyte public key. Encryption and decryption remain unaffected by our optimizations: As an example, the 781-bit variant requires 152 multiplications per encryption.

Keywords: Public key cryptography, NS cryptosystem, multiplicative knapsack, efficiency.

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 $v_i = \sqrt[s]{p_i} \mod p$. 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[3] and denote by n the largest integer such that:

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

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 2048-bit p corresponds to a sixteen kilobyte public-key and allows encrypting 233-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

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

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

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 approaches. Indeed, 233-bit plaintexts are often insufficiently large in practice to include the message, the randomizer (typically 128 bits) and the redundancy (at least 160 bits, or better 256 bits) that one needs to use chosen-ciphertext secure transformations such Fujisaki and Okamoto's [FO99,FO00].

In the next section, we propose a technique based on modular fractions that multiplies bandwidth by $\log_2 3 \simeq 1.58$ for binary-message NS.[4] 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. Final figures are given in Section 4.3. In Section 5, we examine the security of the proposed improvements. Finally, Section 6 describes combinatorial problems whose solutions might yield even more efficient NS variants.

2 Fractional Message Encoding

In this section, we show that using signed message bits allows to increase bandwidth at no cost. 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 recovers a u such that:

$$u = c^s = \frac{a}{b} \mod p, \qquad \text{with} \begin{cases} a = \prod\limits_{m_i=1} p_i \\ b = \prod\limits_{m_i=-1} p_i. \end{cases} \quad \text{and } \gcd(a,b) = 1.$$

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.*

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

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 tackle 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 modular 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 becomes 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 $\log_2 3$.

It is important to underline that while different signed binary representations of a given m exist (*e.g.* $10\underline{1} = 011$ *i.e.* $2^2 - 2^0 = 2^1 + 2^0$), the above procedure will recover the *exact* encoding used to encrypt m and not an equivalent one.

Note that as $\ell > 1$ is used in conjunction with this technique (*i.e.*, in the $(\ell+1)$-base variant), bandwidth improvement tends to one 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

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.[5] 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:

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

[5] For the sake of simplicity, we now define the first prime as $p_1 = 2$.

In other words, we do not request 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:

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

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 $i + 1$.

3.1 A Small Example

We illustrate the mechanism by a small toy-example.

• 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:

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

• ENCRYPTION OF $m = 35$:

We start by writing m is base $\gamma = 4$, *i.e.*, $m = 35 = 203_4$ and encrypt it as:

$$c = v_{(0\cdot4+\mathbf{3}+1)} \times v_{(1\cdot4+\mathbf{0}+1)} \times v_{(2\cdot4+\mathbf{2}+1)} = v_4 \times v_5 \times v_{11} \bmod 4931 = 4484.$$

• DECRYPTION:

By exponentiation, the receiver retrieves:

$c^s \bmod p = 4484^{3079} \bmod 4931 = 7{\times}11{\times}31 = p_{(0\cdot4+\mathbf{3}+1)}{\times}p_{(1\cdot4+\mathbf{0}+1)}{\times}p_{(2\cdot4+\mathbf{2}+1)}$,

whereby $m = 203_4$.

3.2 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 algorithm for a 2048-bit p. The first row represents 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	233	233 bits	59 kilobytes	0.11
2	208	207 bits	104 kilobytes	0.10
4	189	378 bits	189 kilobytes	0.18
8	172	516 bits	344 kilobytes	0.25
16	159	635 bits	636 kilobytes	0.31
32	147	734 bits	1176 kilobytes	0.36
64	137	821 bits	2192 kilobytes	0.40
128	128	896 bits	4096 kilobytes	0.44
256	121	967 bits	7744 kilobytes	0.47
512	114	1025 bits	14592 kilobytes	0.50
1024	108	1080 bits	27648 kilobytes	0.53

As one can see, bandwidth improvement is significant, but the public keys are huge. Fortunately, we address this issue in the Section 4.

3.3 Linear Bandwidth

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

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

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 constant-factor bandwidth gains.

3.4 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 3.1.

Figures are given in the following table, where the first row (*i.e.*, $\gamma = 2$) represents the original NS. As before, this results assume a 2048-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	233	233 bits	59 kilobytes	0.11 (original NS)
4	196	392 bits	147 kilobytes	0.19
8	175	525 bits	307 kilobytes	0.26
16	160	640 bits	600 kilobytes	0.31
32	148	740 bits	1147 kilobytes	0.36
64	137	822 bits	2158 kilobytes	0.40
128	128	896 bits	4064 kilobytes	0.44
256	121	968 bits	7714 kilobytes	0.47
512	114	1026 bits	14564 kilobytes	0.50
1024	108	1080 bits	27621 kilobytes	0.53

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[6] 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:

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

$m_0 = 12$ 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 bandwidth improvement stems 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 as 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[7] and consider n packs containing γ small primes each (the primes start from $p_1 = 2$). We pick a prime p such that:

[6] 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 whose minimal and maximal p_i-s are roughly equivalent in size, the number of products quickly tends to $\binom{\gamma+\ell}{\ell}$.

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

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

As there are[8] 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 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:

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

where $\{d_{i,1}, \ldots, d_{i,\gamma}\} = \mathsf{unrank}(m_i)$.

To decrypt c, the receiver simply factorizes $c^s \bmod p$ in $\mathbb{N}$ and recovers 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 2048-bit p. The first line $\{\gamma, \ell\} = \{1, 1\}$ is the original NS:

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	233	233 bits	59 kilobytes	0.11
8	66	5	169 bits	10 kilobytes	0.08
16	54	5	255 bits	20 kilobytes	0.12
64	73	3	398 bits	48 kilobytes	0.19
512	38	4	781 bits	512 kilobytes	0.38
128	10	16	781 bits	512 kilobytes	0.38

Note that encryption is very fast, since it requires $\ell \cdot n$ multiplications *e.g.* in the 781-bit setting an encryption claims 152 multiplications.

5 Security Considerations

As stressed previously, no security proof is known for the original NS, and we have no hope nor claim that our modifications may supplement this lack. In this section we nonetheless recall certain security-related facts, some of which are already known since [NS97], for the clarity and the self-containment of this paper.

[8] *cf.* to Appendix A for a proof.

5.1 What Security Can Be Attained?

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

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

The original NS cryptosystem, or the variants presented in Sections 2, 3 or 4 can not ensure indistinguishability, since they are by nature deterministic. The hope however is that there might be one-way. To achieve full-security with our variants (or with NS), one can use generic transformations such as [FO99,FO00]: nevertheless, as there are no formal reductions from a standard hard problem to an attack of NS-type schemes (be these the original NS or the variants proposed herein), the application of these generic rules cannot possibly achieve a provably security, but only give empirical security arguments.

5.2 Security Arguments

OUR SCHEMES CAN BE BROKEN IF ONE SOLVES THE DISCRETE-LOGARITHM. It is clear that a discrete-logarithm oracle will 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 is actually the secret key s. Even if the primes are permuted or made secret, the fact that primes must be small makes them easily guessable.

LARGER MESSAGE SPACE MAY MAKE NS-TYPE PROBLEMS HARDER. As one can see, the schemes presented in this paper are — as is the original NS — multiplicative knapsacks. Even if no efficient algorithm solving this problem is known, one must ensure that a brute-force attack consisting in testing all products is impossible. More precisely, getting back the arguments put forward in Section 2.3 of [NS97], the message space must at least exceed 160 bits, if one requires an 80-bit security, or 256 bits, if one wants 128-bit security. In this perspective, our bandwidth improvements indirectly improve security by easing the attainment of a larger message space. However, we cannot claim that the variants are stronger, as bandwidth improvements come along with larger public-keys, which — at least in the information theoretic sense — give more information to the attacker about the secret key.

SMALL FACTORS OF $(p-1)$. As stressed in Section 2.4 of [NS97], the small factors of $(p-1)$ are important. Denote by $\mathbb{QR}_p$ and $\overline{\mathbb{QR}}_p$ the quadratic and non-quadratic residues modulo p respectively. Let

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

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

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

which in turn leaks the value $\sum_{v_i \in \overline{\mathbb{QR}}_p} m_i \bmod 2$. This partial information leakage can also be applied to other small factors of $(p-1)$. Therefore, [NS97] advised to use a strong prime p, and to spare one m_i to compensate the leakage (in other words, they simply make $\sum_{v_i \in \overline{\mathbb{QR}}_p} m_i \bmod 2$ constant).

In our variants, it is not as simple to use same attacks. Indeed, in any given prime pack, one expects to have 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 reveals the parity of the number of message digits m_i whose corresponding primes are in $\overline{\mathbb{QR}}_p$. Even if leakage is less precise than in the NS case, we still recommend the use of a strong prime (and residue value compensation) with our variants.

Can a reduction from attacking NS to attacking our variants exist? At a first glance, it might seem that reductions between NS and our variants exist: indeed, one may hope that access to a decryption oracle $\mathcal{D}$ of one of our schemes would yield an NS decryption oracle $\mathcal{D}'$.

However, a simple observation shows that this is certainly impossible: in our case, the public key is longer and contains more elements related to the secret key. Therefore, from an NS public key and a challenge, it may certainly be possible to build a challenge for our variants, but there is little hope that one might reconstruct the entire public key.

Thus, we have no formal proof that the security of the original NS is equivalent to the security of the variants proposed herein.

6 Further Research

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 $\mathcal{S} = \{S_1, ..., S_n\}$ be n integer-sets, each of size γ and denote by $S_i[j]$

the j-th element of S_i. We call $\mathcal{S}$ an *encoder* if its S_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 $\mathcal{S}$ occur. Improving the NS consists in finding "better" encoders.

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

$$h(\mathcal{S}) = \prod_{i=1}^{n} \max_j \left(S_i[j]\right)$$

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

We saw that when the $S_i[j]$ are the first small primes, $\mathcal{S}$ is an encoder and $h(\mathcal{S}) = \prod_{i=1}^{n} p_{i\gamma}$ (Section 3). We also saw that when the smallest element in each S_i is one, the resulting $\mathcal{S}$ is still an encoder whose head-product is $h(\mathcal{S}) = \prod_{i=1}^{n} p_{i(\gamma-1)}$ (Section 3.4).

This gives raise to interesting combinatorial problems such as finding algorithms for efficiently testing that a given $\mathcal{S}$ 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 $\mathcal{S}$ whose $h(\mathcal{S}) = 4 \times 8 \times 13 = 416$:

$$\text{pack 1}\begin{cases} S_1[1] = 1 \\ S_1[2] = 2 \\ S_1[3] = 3 \\ S_1[4] = 4 \end{cases} \qquad \text{pack 2}\begin{cases} S_2[1] = 1 \\ S_2[2] = 5 \\ S_2[3] = 7 \\ S_2[4] = 8 \end{cases} \qquad \text{pack 3}\begin{cases} S_3[1] = 1 \\ S_3[2] = 9 \\ S_3[3] = 11 \\ S_3[4] = 13 \end{cases}$$

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 $\mathcal{S}$-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 rationals.

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 Computing $\mathcal{R}_{\ell,\gamma}$

In this appendix, we recall 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 $\sum_k d_k \leqslant \ell$.

$\binom{\gamma+i-1}{i}$ is the number of sequences of γ integers whose sum equals 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 stems from Pascal's rule.

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

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