# Speeding-Up Verification of Digital Signatures

Abdul Rahman Taleb, Damien Vergnaud

# Speeding-Up Verification of Digital Signatures

Abdul Rahman Taleb[1], Damien Vergnaud[2,]

**Abstract**

In 2003, Fischlin introduced the concept of progressive verification in cryptography to relate the error probability of a cryptographic verification procedure to its running time. It ensures that the verifier confidence in the validity of a verification procedure grows with the work it invests in the computation. Le, Kelkar and Kate recently revisited this approach for digital signatures and proposed a similar framework under the name of flexible signatures. We propose efficient probabilistic verification procedures for popular signature schemes in which the error probability of a verifier decreases exponentially with the verifier running time. We propose theoretical schemes for the RSA and ECDSA signatures based on some elegant idea proposed by Bernstein in 2000 and some additional tricks. We also present a general practical method, that makes use of efficient error-correcting codes, for signature schemes for which verification involves a matrix/vector multiplication.

*Keywords:* Public-Key Cryptography, Digital Signatures, Probabilistic Verification, RSA Signatures, ECDSA Signatures, GPV Signatures

## 1. Introduction

It is common in public-key cryptography, that parties need to repeatedly verify equations that ensure the validity of some input (*e.g.* for digital signature schemes, commitment schemes, or proof systems). In 2003, Fischlin [21] introduced the idea of *progressive verification* in cryptography to relate the error probability of a verification procedure to its running time. This concept overcomes the property that a verifier remains oblivious about the validity of a verification procedure until it is actually completed. Fischlin's approach ensures that the verifier confidence grows with the work it invests in the computation.

Digital signatures are arguably the most important cryptographic primitive and fast signature verification is extremely desirable in many applications (especially for numerous low computation scenarios such as RFID, wireless sensors or smart cards). Therefore, as explicitly mentioned in Fischlin's work,

*Email addresses:* taleb.abdulrahman1@gmail.com (Abdul Rahman Taleb), damien.vergnaud@lip6.fr (Damien Vergnaud)

[1]Sorbonne Université, CNRS, LIP6, Paris, France and CryptoExperts

[2]Sorbonne Université, CNRS, LIP6, Paris, France and Institut Universitaire de France

it is interesting to find non-trivial signature schemes in which the underlying number-theoretic function is somewhat progressively verifiable. In a digital signature protocol with progressive verification, the goal is to design a probabilistic verification procedure (in addition to the classical deterministic verification algorithm) that takes as input a message $m$, a public-key $\mathsf{vk}$, a putative digital signature $\sigma$ on $m$ for $\mathsf{vk}$, and a timing parameter $\tau$ and outputs some real number $\alpha \in [0, 1]$ (or the special symbol $\perp$ if $\sigma$ is detected as invalid). The progressive verification algorithm never rejects valid signatures and $\alpha$ represents its confidence level on the signature validity. This algorithm can be seen as a probabilistic method to spot flawed signatures faster than using the classical deterministic verification algorithm. The progressive verification procedure is sound if it accepts any invalid signature with probability at most $\alpha$ (over the random choices of the verification procedure). Recently, Le, Kelkar and Kate recently revisited Fischlin's concept for digital signatures (under the name of *flexible signatures*) [41]. They presented a progressive probabilistic verification for the classical one-time signatures of Lamport. The goal of this paper is to present efficient and secure progressive verification procedures for popular signature schemes (including RSA, ECDSA and GPV signature schemes).

### *1.1. Prior Work*

Digital signatures are a cryptographic mechanism used to verify the authenticity and integrity of digital data. They allow a signer who has established a public verification key to sign a message such that any other party can verify that the message originated from the signer and was not modified in any way. This primitive is of paramount importance for building secure systems and are used in most real-world security protocols. However, for most signature schemes, many computation devices with limited computation power are not able to perform signature verification which is both time and power consuming. Many techniques have been suggested over the years to decrease the computational overhead of signature verification.

A lot of work in cryptography has been devoted to design schemes allowing to perform expensive tasks in batch rather than individually to achieve better efficiency. For instance, in order to speed up the verification of several signatures, Bellare, Garay and Rabin [4] proposed *batch verification* to securely verify a set of digital signatures. For the verification of individual signatures, the concept of *server-aided verification* was proposed by Quisquater and de Soete [45] for speeding up RSA verification with a small exponent. Protocols for signature schemes were subsequently proposed and Girault and Lefranc formalized the concept in [29]. This paradigm allows signatures to be verified by executing an interactive protocol with an untrusted server which requires less computation than the original verification algorithm of the digital signature.

For all these verification procedures, a verifier performs a certain number of verification steps and finally outputs a decision; the error probability of this decision is usually null or at most negligible. However, if a verification algorithm is stopped after some computation steps but before its end, the algorithm usually cannot predict the result better than before the start of computation.

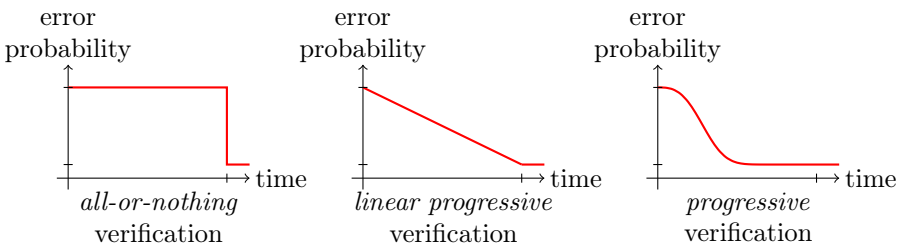


Figure 1: Principle of Progressive Verification

In [21], Fischlin called this *all-or-nothing verification*: in order to give a reliable decision one must run the full verification procedure (see Figure 1, left part). Fischlin introduced the idea of *progressive verification* with the goal to relate the error probability of the decision to the running time of the verifier. As shown in Figure 1 (right part), one wants the error probability of the verifier's decision to decrease quickly with the number of performed steps of the verification procedure. Fischlin focused on message authentication codes in [21] and hash chains in [22] and left open the problem of designing signature schemes in which the underlying number-theoretic function is progressively verifiable.

As mentioned above, Le, Kelkar and Kate [41] showed that the classical one-time Lamport signature scheme [40] is suitable for progressive verification in which the error probability of the verifier's decision decrease *linearly* with the number of performed steps of the verification procedure. Indeed, in this signature scheme, a verifier validates a signature by checking whether $k$ (independent) equations hold (where $k$ is the length of the signed message), and the progressive verification procedure consists in checking each equation in a random order. In the following, we will use the term *linear progressive verification* for such schemes that run atomic verification of independent computation in a random order and for which the error probability thus decreases linearly with its number of sequential executions (see Figure 1, middle part). They also proposed a (linear) progressive verification scheme for Merkle tree signatures [43]. This idea of using intermediate values of the computation has also been used in the context of verification of hash chains before (see e.g. [34, 35, 36, 37] and references therein).

Many identification protocols proceeds in sequential rounds such that in each round a cheating prover will be caught with some small probability, say, 1/2 or 2/3 (*e.g.* [19] in factoring-based cryptography, [49, 50] in code-based cryptography, [39] in lattice-based cryptography or [46] in multivariate cryptography). Repeating the protocol permits to reduce the soundness error of these identification systems. Fischlin already noted in [21] that using the Fiat-Shamir heuristic [20], we obtain digital signature schemes suitable for linear progressive verification. These resulting signature schemes are often less efficient (in computational complexity or communication complexity) than other schemes based on

similar assumptions, but which do not support the linear progress of verification (*e.g.* [32] in factoring-based cryptography, [16] in code-based cryptography, [42] in lattice-based cryptography and [44] in multivariate cryptography). The latter schemes often significantly outperform the former ones even with probabilistic verification.

In their paper, Le *et al.* mentioned that lattice-based signature schemes such as GPV signatures [27] are also good candidates for progressive verification. Indeed, for this scheme, the public verification key is a matrix, a signature is a vector and verification consists in a matrix/vector multiplication. Le *et al.* outlined a progressive verification method in which one checks the validity on each row of the matrix (in a random order) but again the error probability of the verifier's decision decreases only linearly with the verifier computation time. Finally, they also claimed that "*prominent traditional signature schemes such as RSA, (EC)DSA seem unsuitable towards building flexible signatures*".

### *1.2. Contributions of the Paper*

We revisit the concept of progressive verification for digital signatures, with the goal to propose efficient probabilistic verification procedures for popular signature schemes in which the error probability decreases exponentially with the number of verification steps (see Figure 1, right part). We present three techniques for such progressive verification of important digital signature schemes based on the RSA primitive, on the discrete logarithm problem on elliptic curves (and in particular ECDSA) and on GPV signatures and variants where the verification procedure consists in a matrix/vector multiplication.

#### *1.2.1. RSA signatures*

The RSA signature is one of the most elegant and well-known signature schemes. It is extensively used in a wide variety of applications, and serves as the basis of several existing standards. The public key is a pair $(N, e)$ where $N$ is an integer (of unknown factorization) and $e$ is an integer coprime with $\varphi(N)$ (the Euler totient function of $N$); a signature of a message $m$ is an element $\sigma \in \mathbb{Z}_N$ such that

$$\sigma^e \equiv \mathcal{H}(m) \bmod N \tag{1}$$

where $\mathcal{H}$ is an appropriate hash function that maps messages of arbitrary length to elements in $\mathbb{Z}_N$.

In [6], a paper that predates Fischlin's paper [21], Bernstein proposed an elegant verification procedure for Rabin signatures which can be seen as probabilistic verification. In Rabin signatures, the verification equation is roughly speaking (1) but with $e = 2$; in this case, one can add to the signature the integer $k$ such that $\sigma^2 = \mathcal{H}(m) + kN$ (and adding this element does not decrease the security since it can be computed easily from the signature $\sigma$). Using classical techniques, one can then check the validity of this equation over $\mathbb{Z}$ by mapping it to a random quotient ring $\mathbb{Z}_p$ of $\mathbb{Z}$ for some prime number $p$. To increase its confidence, a verifier can simply map the equality to enough quotient rings and the error probability decreases exponentially with the number of tested prime

numbers. Since, in this case, we have $e = 2$ in (1), the integer $k$ is smaller than $N$ and the signature size is at most doubled. A naive approach for RSA signatures (already mentioned by Wagner in [53]), is to add to the signature the integer $k$ such that $\sigma^e = \mathcal{H}(m) + kN$ and then check this equality modulo small prime numbers. This approach is only suitable for very small public exponents $e$ since the integer $k$ is close to $N^{e-1}$ and thus the signature size is multiplied by $e$. As a first example of a signature scheme with progressive verification, we present a variant of the RSA signature based on Bernstein's technique where the signature size is multiplied only by a factor $O(\log(e))$. It relies on a simple trick and serves as an introduction to the primitive. It is worth mentioning that this verification technique to the so-called RSA Full Domain Hash signature scheme [5] can also be applied to other paddings (*e.g.* RSA-PSS [5]) and variants of RSA in the standard security model (i.e. without the random oracle heuristic assumption, *e.g.* Cramer-Shoup signatures [14]). This construction is mainly of theoretical interest since, even if it achieves a good efficiency improvement in the verification procedure, the increase of the signature size makes it useless in most practical settings where the additional power consumption for reading the signature outweighs the efficiency gain.

*1.2.2. ECDSA signatures*

The Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) [38] are efficient digital signature schemes which produce (reasonably) short signatures and whose securities rely on the discrete logarithm problem in their respective underlying groups. Due to its small key size, thanks to the (assumed) stronger difficulty of this problem on elliptic curves, ECDSA has a fast signing algorithm and is therefore used in many standard cryptographic libraries and in cryptocurrencies. For a security parameter $\lambda$, ECDSA signatures have size $O(\lambda)$ bits and their verification requires $O(\lambda)$ operations in the underlying finite field $\mathbb{F}_q$ with $\log(q) = O(\lambda)$. This gives an overall binary complexity $O(\lambda^3)$ (*i.e.* without using fast arithmetic[3] in $\mathbb{F}_q$) for verification which is asymptotically faster than the $O(\log(e)\lambda^6)$ binary complexity for RSA verification (for which the key size is cubic in the security parameter). However, in practice for $\lambda = 128$ and small public exponent $e$, RSA verification is faster than ECDSA verification. It is thus interesting to propose an efficient progressive verification procedure for ECDSA.

A natural approach is to use Bernstein's idea from [6] and to consider all equalities that occur in an ECDSA verification over the ring $\mathbb{Z}$ instead of over $\mathbb{Z}_q$. It is indeed possible to combine readily this idea with our trick introduced for RSA progressive verification for models of elliptic curves where the group arithmetic does not involve division in the ring $\mathbb{Z}_q$. These elliptic curve models are the most widely used in cryptography since it is usually too costly to compute inverses in $\mathbb{Z}_q$. We present a more efficient approach using the clas-

[3]Using fast arithmetic, the verification algorithms have bit complexity $\tilde{O}(\lambda^2)$ for ECDSA and $\tilde{O}(\log(e)\lambda^3)$ for RSA (where the $\tilde{O}$ notation "ignores" logarithmic factors).

sical Weierstrass affine model of elliptic curves. In this case, the group law on the elliptic curve requires the computation of inverses modulo $q$ and it is not clear how to handle these equations in the ring $\mathbb{Z}$ and its quotient rings $\mathbb{Z}_p$ for small prime numbers $p$. Indeed, for a small prime $p$, a random element in $\{1, \ldots, q-1\}$ will not be invertible modulo $p$ with probability close to $1/p$. We present a progressive verification scheme for ECDSA signature with bit complexity $O(\tau \log(p)\lambda^2)$ where the probability error decreases exponentially with $\tau$, and $log(p)$ is much smaller than $\lambda$. The signatures generated have size $O(\lambda^2)$ bits (e.g. still asymptotically shorter than RSA signatures) and their security is identical to the security of original ECDSA signatures. As for RSA, our technique is generic and can be applied to many elliptic-curve based digital signatures (*e.g.* Schnorr signatures [47]) and is also mainly of theoretical interest (since the increase of the signature size outweighs the efficiency gain).

*1.2.3. GPV signatures and variants*

In 2008, Gentry, Peikert and Vaikutanathan [27] proposed a lattice-based signature scheme with strong security guarantees and good asymptotic efficiency. Given some integers $p$, $n$, $m$, the public key is a basis $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ of a random lattice $\Lambda$; a signature of a message $m$ is a "short" vector $\vec{\sigma} \in \mathbb{Z}_p^m$ such that:

$$\mathbf{A} \cdot \vec{\sigma} = \mathcal{H}(m) \tag{2}$$

where $\mathcal{H}$ is an appropriate hash function that maps messages of arbitrary length to elements in $\mathbb{Z}_p^n$. Checking that $\vec{\sigma}$ is "short" can be done efficiently and the most time-consuming operation in GPV signature verification is checking whether (2) holds. As mentioned above, it consists therefore in performing a matrix/vector multiplication in $\mathbb{Z}_p$ and one can obtain a linear progressive verification procedure by checking each row of (2) in a random order.

As the main contribution of the paper, we propose a progressive verification scheme for GPV signatures. In randomized computation, Freivalds' algorithm [24] is a well-known probabilistic algorithm used to verify (square) matrix multiplication: given three matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ in $\mathbb{Z}_p^{n \times n}$, the algorithm checks whether $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ in time[4] $O(\tau n^2)$ for some parameter $\tau$ (with error probability bounded by $2^{-\tau}$). It simply picks uniformly at random $\tau$ vectors $\vec{r_1}, \ldots, \vec{r_\tau} \in \mathbb{Z}_p^n$ and computes successively $\vec{s_i} = \mathbf{B}\vec{r_i}$, $\vec{t_i} = \mathbf{A}\vec{s_i} = (\mathbf{A} \cdot \mathbf{B})\vec{r_i}$ and $\vec{u_i} = \mathbf{C}\vec{r_i}$ for $i \in \{1, \ldots, \tau\}$. and accepts if and only if $\vec{t_i} = \vec{u_i}$ for all $i \in \{1, \ldots, \tau\}$. This approach does not seem applicable to check our matrix/vector multiplication (at least without doing some off-line pre-computation[5]).

[4] *i.e.* faster than computing $AB$ with requires $O(n^{2.3729})$ operations in $\mathbb{Z}_p$ using the best known matrix multiplication algorithm [54].

[5] **Added in proof.** Recently, Boschini, Fiore and Pagnin [11] revisited flexible signatures from [41] . They proposed another way to speed up the verification process of lattice-based signatures (similar to Freivalds' algorithm [24]). It requires (secret) pre-processing from the verifier (that can be re-used an unbounded number of times). The verifier has to store some secret verification key (different for each signer public key) but the resulting verification procedure is then very efficient.

Instead, our approach makes use of error-correcting codes and in particular of $(\ell, n, \delta\ell)$-linear codes which minimum distance satisfies the Gilbert-Varshamov Bound. The new progressively verifiable signature scheme does not modify the signatures (and in particular does not increase their size) but modifies the public key (and increases its size). We consider $\mathbf{G} \in \mathbb{Z}_p^{\ell \times n}$ a generator matrix for some linear code over $\mathbb{Z}_p$ with $\ell > n$. The new public key consists in the two matrices $\mathbf{G} \cdot \mathbf{A} \in \mathbb{Z}_p^{\ell \times m}$ and $\mathbf{G} \in \mathbb{Z}_p^{\ell \times n}$. Instead of checking whether (2) holds, the verification procedure checks the equality

$$(\mathbf{GA}) \cdot \vec{\sigma} = \mathbf{G} \cdot \mathcal{H}(m) \tag{3}$$

which obviously holds if (2) holds (and conversely (2) holds if (3) holds and $\mathbf{G}$ is of full rank). This approach makes the complete verification of (3) costlier than the one of (2) but allows for progressive verification. Indeed, if the code generated by $\mathbf{G}$ is a "good" error-correcting code then it has no non-zero codewords of "small" Hamming weight (i.e. with a "large" number of null coordinates in $\mathbb{Z}_p$). Therefore, if $\mathbf{A} \cdot \vec{\sigma} - \mathcal{H}(m) \neq \vec{0}$, then the vector $\mathbf{G} \cdot (\mathbf{A} \cdot \vec{\sigma} - \mathcal{H}(m))$ has a "large" number of non-zero coordinates and checking that a "sufficiently large" set of rows in (3) coincide ensures with "good" probability that (3) holds for all rows (and thus that the signature is valid). Using reliable and probabilistic constructions of $(\ell, n, \delta\ell)$-linear codes, we can make all these statements mathematically precise and improve significantly the efficiency of GPV signature verification. This approach is actually very general and can be used to improve the efficiency of other cryptographic operations which involve a matrix/vector multiplication. In particular, we show how our technique improves also the verification procedure of the recent signature scheme called Wave and proposed by Debris-Alazard, Sendrier and Tillich in [16].

### *1.3. Organization of the Paper*

The rest of the paper is organized as follows. In Section 2, we briefly describe the model assumed in our paper and provide some notation used throughout it. Section 3 presents and analyzes our RSA-based progressively verifiable signature scheme based on Bernstein's idea from [6]; it also contains efficiency analysis and implementation results. In Section 4, we recall some standard facts on elliptic curve arithmetic and we present our variant of ECDSA signatures with progressive verification; it ends with efficiency analysis of our new scheme. Section 5 recalls needed facts on GPV and Wave signature schemes, on the probabilistic construction of $(\ell, n, \delta\ell)$-linear codes suitable for our needs that satisfies the Gilbert-Varshamov Bound, and presents the new verification scheme together with a mathematical analysis of its security and efficiency. This section ends with a brief performance analysis and comparison to basic verification procedure for these two signature schemes. We conclude the paper with a few open problems for future research in Section 6.

## 2. Preliminaries

### *2.1. Notations*

We denote the security parameter by $\lambda \in \mathbb{N}$ which is given to all algorithms in the unary form $1^\lambda$. Algorithms are randomized unless otherwise stated, and PPT stands for "probabilistic polynomial-time," in the security parameter. We denote random sampling from a finite set $X$ according to the uniform distribution with $x \xleftarrow{\$} X$. We also use the symbol $\xleftarrow{\$}$ for assignments from randomized algorithms, while we denote assignment from deterministic algorithms and calculations with the symbol $\leftarrow$. If $n$ is an integer, we write $\mathbb{Z}_n$ for the ring $\mathbb{Z}/n\mathbb{Z}$. We let $\mathbb{Z}_n^*$ the invertible elements of $\mathbb{Z}_n$. As usual, $f \in \mathrm{negl}(\lambda)$ denotes a function that decreases faster than the inverse of any polynomial in $\lambda$; such functions are called negligible.

In the following, implementation and timings are provided for the different signature schemes. All implementations were run on a laptop computer (Intel(R) Core(TM) i7-8550U CPU, 1.80GHz with 4 cores) using Ubuntu operating system and various cryptographic libraries.

### *2.2. Progressive Verification of Signatures*

In this section, we recall the syntax of digital signatures and the notion of (strong) existential unforgeability. Note that all schemes used in this work are proven existentially unforgeable under standard complexity assumptions (in the random oracle model or in the generic group model). Formally, a digital signature scheme consists of three algorithms, namely the *key generation* algorithm, the *signing* algorithm and the *verification* algorithm.

**Definition 1** (Digital signatures)**.** *A digital signature scheme* $\Sigma$ *is a triple of PPT algorithms* $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *such that:*

- *The* key generation algorithm $\mathsf{KeyGen}$*, on input security parameter* $1^\lambda$*, generates a key pair* $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*;* $\mathsf{sk}$ *is called the* (secret) signing key *and* $\mathsf{vk}$ *is called the* (public) verification key*;*
- *the* signing algorithm $\mathsf{Sign}$*, on input a signing key* $\mathsf{sk}$ *and a message* $m \in \{0,1\}^*$*, outputs a signature* $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, m)$*;*
- *the (deterministic) verification algorithm* $\mathsf{Verify}$*, on input a verification key* $\mathsf{vk}$*, a message* $m \in \{0,1\}^*$*, and a bit string* $\sigma$*, outputs a bit* $b \leftarrow \mathsf{Verify}(\mathsf{vk}, m, \sigma)$*,*

*and the scheme* $\Sigma$ *satisfies the following* correctness *property:* $\forall \lambda \in \mathbb{N}$, $\forall (\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $\forall m \in \{0,1\}^*, \forall \sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, m)$*, then* $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$*.*
*If for some verification key* $\mathsf{vk}$*, some message* $m \in \{0,1\}^*$ *and some bit-string* $\sigma \in \{0,1\}^*$*, we have* $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$ *then* $\sigma$ *is called a* signature on $m$ for $\mathsf{vk}$*.*

**Remark 1.** *It is worth noting that in most existing works, the verification algorithm is assumed to be deterministic. We use the same approach in this work even if our definitions and techniques can be applied to digital signature schemes with probabilistic verification as defined in [23].*

We recall the strong[6] EUF-CMA security notion:

**Definition 2** (Strong EUF-CMA Security)**.** *A digital signature scheme* $\Sigma = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Verify})$ *is* strongly secure against existential forgeries in a chosen-message attack *(strongly* EUF-CMA*-secure) if the advantage of any PPT adversary* $\mathcal{A}$ *against the* EUF-CMA *game defined in Figure 2 is negligible:*

$$\mathsf{Adv}^{\mathsf{EUF}}_{\mathcal{A},\Sigma}(\lambda) = \Pr\left[\mathsf{EUF}^{\mathcal{A}}_{\Sigma}(\lambda) = 1\right] \in \mathrm{negl}(\lambda).$$

$\underline{\mathsf{EUF}^{\mathcal{A}}_{\Sigma}(\lambda)}$:
$L \leftarrow \emptyset$
$(\mathsf{sk}, \mathsf{vk}) \xleftarrow{\$} \Sigma.\mathsf{KeyGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathrm{Sign}(\cdot),\mathrm{Verify}(\cdot,\cdot),H(\cdot)}(1^\lambda)$
if $(m^*, \sigma^*) \notin L$
  return $\Sigma.\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*)$
return 0

$\underline{\mathrm{Sign}(m)}$:
$\sigma \xleftarrow{\$} \Sigma.\mathsf{Sign}(\mathsf{sk}, m)$
$L \leftarrow L \cup \{(m, \sigma)\}$
return $\sigma$

$\underline{\mathrm{Verify}(m, \sigma)}$:
return $\Sigma.\mathsf{Verify}(\mathsf{vk}, m, \sigma)$

Figure 2: Strong EUF-CMA experiment for digital signature schemes.

Given two functions $t : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to [0, 1]$, we also define a $(t, \epsilon)$-EUF-CMA-adversary as a probabilistic algorithm $\mathcal{A}$ for which the expected running time of the experiment $\mathsf{EUF}^{\mathcal{A}}_{\Sigma}(\lambda)$ is upper-bounded by $t(\lambda)$ and its advantage $\mathsf{Adv}^{\mathsf{EUF}}_{\mathcal{A},\Sigma}(\lambda)$ is lower-bounded by $\epsilon(\lambda)$.

We now give the syntax of digital signatures with probabilistic verification. Note that our definition differs from the definition of flexible signatures from [41]. Our security definition for the progressive verification is statistical rather than computational in [41]. For that reason and to avoid confusion between the two primitives, we use the name put forth by Fischlin in his seminal work [21].

**Definition 3** (Digital signatures with progressive verification)**.** *A* digital signature scheme with progressive verification $\Sigma$ *is a* 4*-tuple of PPT algorithms* $\Sigma = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Verify}, \mathit{ProgVerify})$ *such that:*

- $\Sigma = (\mathit{KeyGen}, \mathit{Sign})$ *is a correct digital signature scheme;*

[6] In contrast to the *weak* version, the adversary is allowed to forge for a message that they have queried before, provided that their forgery is *not* an oracle response.

- *the (probabilistic) progressive verification algorithm* $\mathsf{Verify}$*, on input a verification key* $\mathsf{vk}$*, a message* $m$*, and a bit string* $\sigma$*, and some timing parameter* $\tau$ *outputs* $\alpha \xleftarrow{\$} \mathsf{ProgVerify}(\mathsf{vk}, m, \sigma, \tau)$ *where* $\alpha$ *is either a real number in* $[0,1]$ *or the special symbol* $\perp$*,*

*with the following properties:*

1. *if for some verification key* $\mathsf{vk}$*, some message* $m \in \{0,1\}^*$*, some bit-string* $\sigma \in \{0,1\}^*$*, and some timing parameter* $\tau$*,* $\mathsf{ProgVerify}(\mathsf{vk}, m, \sigma, \tau)$ *outputs* $\perp$ *then we have* $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 0$ *(*i.e. *$\sigma$ is not a signature on $m$ for* $\mathsf{vk}$*).*
2. *if for some verification key* $\mathsf{vk}$*, some message* $m \in \{0,1\}^*$*, some bit-string* $\sigma \in \{0,1\}^*$ *and some timing parameter* $\tau$*,* $\mathsf{ProgVerify}(\mathsf{vk}, m, \sigma, \tau)$ *outputs* $\alpha$ *then* $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 0$ *with probability at most* $1 - \alpha$ *(taken over the random coins of* $\mathsf{ProgVerify}$*).*

With our definition, the progressive verification algorithm never rejects valid signatures. When this algorithm $\mathsf{ProgVerify}(\mathsf{vk}, m, \sigma, \tau)$ outputs some real number $\alpha \in [0,1]$, $\alpha$ represents its confidence level on the signature validity: when $\alpha = 0$, the progressive verification algorithm has no information on the validity of $\sigma$ as a signature of $m$ for $\mathsf{vk}$ whereas for $\alpha = 1$, we have $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$ with certainty. This model of progressive verification with "one-sided error" is reminiscent of the probabilistic complexity class $\mathcal{RP}$ and Monte-Carlo algorithms[7]. We can see this definition as a method to spot flawed signatures prematurely. As mentioned in the introduction, a digital signature scheme with progressive verification is interesting if it is secure against existential forgeries in a chosen-message attack and if rejecting invalid signatures grows with the performed work. In the schemes we present in the rest of the paper, we will obtain constructions where $\alpha = 2^{-\Omega(\tau)}$ (*i.e.* in which the error probability decreases exponentially with the timing parameter).

## 3. RSA Signatures

### *3.1. Description*

In this paragraph, we first recall the description of the classical RSA Full Domain Hash (RSA-FDH) signature scheme from [5]. It achieves EUF-CMA-security in the random oracle model assuming the so-called RSA computational assumption [5] (which is a stronger assumption than the difficulty of the integer factoring problem).

Our description involves a security parameter $\lambda \in \mathbb{N}$ (following Definition 1); this integer $\lambda$ represents the bit size of the RSA moduli generated but not

[7] Another possible approach is to consider signatures with progressive verification in which the progressive verification algorithm takes as input a confidence level and prematurely terminate the verification according to this level and then decide about validity of the given signature.

actually the security level of the signature scheme. To achieve a security level of $\kappa$ security bits (*i.e.* a probabilistic adversary cannot break the RSA assumption and thus the scheme in expected time less than $O(2^\kappa)$), we must choose at least $\lambda = \Omega(\kappa^3)$ to prevent the best known factoring methods.

The RSA-FDH signature scheme consists of the following procedures:

- $\mathsf{KeyGen}(1^\lambda)$: In order to generate the keys, one picks uniformly at random two $(\lambda/2)$-bits prime numbers $p$ and $q$, and defines the public modulus $N = pq$. Then, a public exponent $e$ is chosen such that $1 < e < \varphi(N)$ and $\gcd(e, \varphi(N)) = 1$, where $\varphi$ denotes Euler's totient function. The private key $d$ is then computed as the modular inverse of $e$ in $Z_{\varphi(N)}$. The verification key is $\mathsf{vk} = (N, e, \mathcal{H})$ where $\mathcal{H}$ is a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_N$ , while the signing key is $\mathsf{sk} = (N, d, \mathcal{H})$.

- $\mathsf{Sign}(\mathsf{sk}, m)$: To sign a message $m \in \{0,1\}^*$, the signer computes the hash value $h = \mathcal{H}(m) \in \mathbb{Z}_N$, and outputs the signature $\sigma = h^d \bmod N$.

- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: To determine the validity of a signature $\sigma$ of a message $m$, one computes $h' = \sigma^e \mod N$ and $h = \mathcal{H}(m)$. The signature is accepted if and only if $h = h'$.

In this deterministic verification algorithm, the value $h' = \sigma^e \mod N$ needs to be computed entirely before deciding on the validity of the signature. In fact, the verification is an all-or-nothing algorithm where no information is revealed before the complete exponentiation algorithm is executed. Our goal in the rest of this section is to provide a modified version of the RSA signature scheme where the verification is made progressive.

### *3.2. RSA Signatures with Progressive Verification*

As mentioned in the introduction, our progressive scheme of the RSA-FDH signature uses the elegant idea proposed by Bernstein in [6]. To test whether an integer in $\mathbb{Z}$ is equal to 0, one simply tests if it is equal to 0 modulo small prime numbers.

When verifying a signature $\sigma$ of a message $m$, one tests that the equality $\sigma^e = \mathcal{H}(m) \mod N$ holds. This can be rewritten as $\sigma^e - \mathcal{H}(m) = 0 \mod N$, which holds modulo an integer $N$ of bit-length $\lambda$. With the trick from [6], a faster way to test the equality is to consider the quotient $k$ of the Euclidean division of $\sigma^e$ by $N$, and verify that $c = 0$ where $c = \sigma^e - \mathcal{H}(m) - k \cdot N$. This can be done fast by checking if $c$ is divisible by a random prime $p$ of bit-length equal to some parameter $\mu \ll \lambda$. Indeed, since $c$ is an integer of bit-length $e \cdot \lambda$, it cannot be divisible by more than $\lceil e \cdot \lambda / \mu \rceil$ distinct primes of bit-length $\mu$. By the prime number theorem (see below for a precise statement), it is known that the number of prime numbers of bit-length $\mu$ is roughly equal to $2^\mu/(2\mu)$. For a random prime $p$ smaller than $2^\mu$, we thus have :

$$\Pr_{p \text{ prime},\ 2^{\mu-1}<p<2^\mu}[c \neq 0 \wedge c = 0 \mod p] \leq \frac{e \cdot \lambda}{2^{\mu-1}} \tag{4}$$

This is the probability of accepting an invalid signature, when $c = \sigma^e - \mathcal{H}(m) - k \cdot N \neq 0$ but is divisible by the prime $p$ which makes the equality $c = 0 \mod p$ hold. For large $\mu$, this probability becomes negligible, and for progressive verification one can choose $\tau$ different random primes of length $\mu$ uniformly at random instead of one, to make this probability even smaller (namely upper-bounded by $(e\lambda/2^{\mu-1})^\tau$). If we consider for example RSA-2048 (*i.e.* with $\lambda = 2048$) with $e = 2^{16} + 1$, choosing one prime of length $\mu = 32$ would make the error probability as small as $2^{-4}$, and choosing $\tau$ primes makes it smaller than $2^{-4\tau}$ (which decreases exponentially with $\tau$).

However, the problem with this method is that the output signature needs to be $(\sigma, k)$, where the bit-length of $k$ is approximatively equal to $(e-1)\lambda$, and for large values of $e$ (typically for $e \approx N$), outputting $k$ in the signature becomes exponential in $\lambda$ and totally impractical. For this reason, in our proposed modified signature scheme, we use an additional trick by extending the signature with all the intermediate values of $\sigma_i$ and $k_i$ which appears in the "square-and-multiply" algorithm performed during the computation of $\sigma^e \mod N$. The modified signature scheme Progressive-RSA is described below. For simplicity, we consider only public exponents $e$ of the form $e = 2^h + 1$ for some parameter $h$ (since these exponents are the most widely used and it makes the presentation simpler). However, our scheme can be readily generalized to any public exponent by considering any addition chain for computing $e$.

- $\mathsf{KeyGen}(1^\lambda)$: The procedure is the same as the key generation for classical RSA-FDH signature. The verification key is $\mathsf{vk} = (N, e, \mathcal{H})$ with $e = 2^h + 1$ for some parameter $h$ and $\mathcal{H}$ is a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_N$, while the signing key is $\mathsf{sk} = (N, d, \mathcal{H})$.

- $\mathsf{Sign}(\mathsf{sk}, m)$: Given a message $m$, the algorithm first computes the classical RSA-FDH signature $\sigma_0 = \mathcal{H}(m)^d \mod N$. It then computes $\sigma_0^e \mod N$ using the "square-and-multiply" algorithm, and stores the $(2h+1)$ integers $k_0$ and $\{\sigma_i, k_i\}_{i\in[h]}$, where $k_i$ and $\sigma_i$ are respectively the quotient and remainder of the Euclidean division of $\sigma_{i-1}^2$ by $N$ for $i \in [h]$ and $k_{h+1}$ is the quotient of the Euclidean division of $\sigma_0 \cdot \sigma_h$ by $N$ . The algorithm outputs the signature $\sigma = (\sigma_0, \left\{\sigma^{2^i}, k_i\right\}_{i\in[h+1]}, k_{h+1})$.

- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: A signature $\sigma = (\sigma_0, \left\{\sigma^{2^i}, k_i\right\}_{i\in[h]}, k_{h+1})$ of a message $m$ is valid, if and only if $\sigma_{i-1}^2 - \sigma_i - k_i \cdot N = 0$ for all $i \in [h]$ and $\sigma_0 \cdot \sigma_h^2 - \mathcal{H}(m) - k_{h+1} \cdot N = 0$.

In this scheme, each of the $\sigma_i$ and $k_i$ are of bit-length $\lambda$ at most. The signature length is then roughly $2(h+1)\lambda$. The verification algorithm is slightly faster than the verification algorithm of the RSA-FDH scheme (since it removes the Euclidean divisions and only requires ring operations in $\mathbb{Z}$). The signing algorithm is slightly slower since it involves running the verification algorithm of RSA-FDH (but for $e = 2^{16}+1$ for instance, the increase is marginal). The verification algorithm of this new scheme only involves ring operations in $\mathbb{Z}$ and one

can apply Bernstein's idea from [6] to obtain Progressive-RSA($\mu$) a progressive verification scheme for RSA signatures (parameterized by some integer $\mu$):

- ProgVerify($\mathsf{vk}, m, \sigma, \tau$): Given a putative signature $\sigma$ of a message $m$ for $\mathsf{vk}$ parsed as $\sigma = (\sigma_0, \left\{\sigma^{2^i}, k_i\right\}_{i\in[h]}, k_{h+1})$, the algorithm picks uniformly at random a prime number $p_j$ of bit-length $\mu$, sequentially for $j \in [\tau]$, and checks whether $\sigma_{i-1}^2 - \sigma_i - k_i \cdot N = 0 \bmod p_j$ for all $i \in [h]$ and $\sigma_0 \cdot \sigma_h^2 - \mathcal{H}(m) - k_{h+1} \cdot N = 0 \bmod p_j$ for $j \in [\tau]$. If one equation does not hold, the algorithm returns $\perp$; otherwise it returns $\alpha = 1 - \left(\frac{\lambda}{2^\mu}\right)^\tau$.

The advantages of the verification in this scheme, apart from the fact that it becomes progressive, is that the operations are all modulo primes of size much smaller than that of $N$. For a fixed error probability, there is clearly a trade-off between the size $\mu$ of the primes to generate and their number $\tau$.

*3.3. Security Analysis*

The following proposition claims that the previous digital signature scheme Progressive-RSA is not less secure than the RSA-FDH classic scheme:

**Proposition 1.** *Let $t : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to [0, 1]$, be two functions and let $\mathcal{A}$ be $(t, \epsilon)$-*EUF-CMA*-adversary against Progressive-RSA. There exist $\mathcal{B}$ a $(t', \epsilon)$-*EUF-CMA*-adversary against RSA-FDH with $t' = t + q_S \cdot T_{\text{RSA-FDH.Verify}}$ where $T_{\text{RSA-FDH.Verify}} : \mathbb{N} \to \mathbb{N}$ denotes the running time of the RSA-FDH.Verify (*i.e. *$T_{\text{RSA-FDH.Verify}}(\lambda) = O(\lambda^3)$*) and $q_S$ denotes the number of queries made by $\mathcal{A}$ to its signing oracle.*

*Proof.* The proof is straightforward. As mentioned above, the signing algorithm Progressive-RSA.Sign first runs RSA-FDH.Sign to obtain $\sigma_0 = \mathcal{H}(m)^d \mod N$ for a message $m$ and then runs the algorithm RSA-FDH.Verify (with no secret input) to compute the full signature $\sigma = (\sigma_0, \left\{\sigma^{2^i}, k_i\right\}_{i\in[h+1]}, k_{h+1})$. Given a public key for the scheme Progressive-RSA, the algorithm $\mathcal{B}$ simply forwards it to the algorithm $\mathcal{A}$. For each signing query made by $\mathcal{A}$ on a message $m_i$ for $i \in [q_S]$, $\mathcal{B}$ forwards it to its own signing oracle to obtain $\sigma_0^{(i)} = \mathcal{H}(m_i)^d \mod N$ for $i \in [q_S]$. The algorithm $\mathcal{B}$ runs RSA-FDH.Verify on each triple $(\mathsf{vk}, m_i, \sigma_0^{(i)})$ for $i \in [q_S]$ to complete the full signature computation. Eventually $\mathcal{A}$ outputs a forgery $(m^\star, \sigma^*)$ valid with probability at least $\epsilon(\lambda)$ and $\mathcal{B}$ can simply extracts a forgery for $m^\star$ for RSA-FDH as $\sigma_0^\star$ if $\sigma^\star$ is parsed as $\sigma^\star = (\sigma_0^\star, k_0^\star, \left\{\sigma^{2^i\star}, k_i^\star\right\}_{i\in[h]})$. □

Following the probabilistic analysis outlined above and using an explicit version of the prime number theorem, we can also easily show that the progressive verification is correct:

**Theorem 1.** *Let $\mu \in \mathbb{N}$ with $\mu > 5$. The digital signature scheme with progressive verification Progressive-RSA($\mu$) is* correct *(*i.e. *it returns $\perp$ only for invalid signatures and if it returns a real number $\alpha \in [0, 1]$ on input $(\mathsf{vk}, m, \sigma, \tau)$ then $\sigma$ is not a signature of $m$ for $\mathsf{vk}$ with probability at most $1 - \alpha$).*

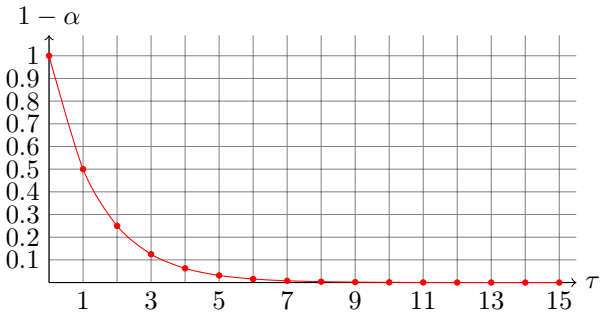


Figure 3: Error probability for 2048-bit RSA signatures with $\mu = 12$

*Proof.* For an integer $x$, we denote as usual $\pi(x)$ the number of primes smaller or equal to $x$. For $x \geq 30$, an explicit version of the prime number theorem [33, 15] proved by Dusart [17, p. 9]) ensures that $0.92x/\ln(x) < \pi(x) < 1.11x/\ln(x)$. Therefore, if $\mu > \log_2 30 + 1$, $\pi\left(2^{\mu}\right) > 0.9 \cdot 2^{\mu}/(\mu \ln(2))$ and $\pi\left(2^{\mu-1}\right) < 1.11 \cdot 2^{\mu}/2(\mu-1)\ln(2)$. The number of primes in $[\![2^{\mu-1}, 2^{\mu}-1]\!]$ is then at least $2^{\mu}/2(\mu-1)$.

Given a putative signature $\sigma$ of a message $m$ for $\mathsf{vk}$ parsed as

$$\sigma = (\sigma_0, \left\{\sigma^{2^i}, k_i\right\}_{i\in[h]}, k_{h+1}),$$

$\sigma$ is valid if and only if the following two conditions hold:

- $\sigma_{i-1}^2 - \sigma_i - k_i \cdot N = 0$ for all $i \in [h]$;
- $\sigma_0 \cdot \sigma_h^2 - \mathcal{H}(m) - k_{h+1} \cdot N = 0$.

Therefore, if $\sigma$ is not valid, at least one equality out of these $(h+1)$ does not hold over $\mathbb{Z}$. This inequality involves integers smaller than $N^2$ and as mentioned above its reduction in a quotient ring $\mathbb{Z}_p$ for some prime number $p$ is an equality for at most $\lceil 2\lambda/\mu \rceil$ primes of bit-length $\mu$. Therefore, for each prime $p_i$ picked uniformly at random among all prime numbers of bit-length $\mu$, the probability that the $(h+1)$ relations hold modulo $p_i$ if $\sigma$ is invalid is upper-bounded by

$$\left\lceil \frac{2\lambda}{\mu} \right\rceil \cdot \left( \frac{2^{\mu}}{2(\mu-1)} \right)^{-1} = \frac{\mu-1}{\mu} \cdot \lambda \cdot 2^{-\mu} \leq \lambda \cdot 2^{-\mu}.$$

Since the algorithm $\mathsf{ProgVerify}$ uses $\tau$ distinct primes of bit-length $\mu$, the probability that an invalid signature is not detected by the verification algorithm is upper-bounded by $(\lambda \cdot 2^{-\mu})^{\tau}$ and thus the real $\alpha = 1 - .(\lambda \cdot 2^{-\mu})^{\tau}$ output by the algorithm provides a correct confidence level for this signature scheme. □

It is worth mentioning that one can indeed conclude with certainty that the classical deterministic verification algorithm would also accept or reject a signature if one runs the progressive verification procedure modulo primes $p_1, ...p_{\ell}$

such that $\prod_{i=1}^{\ell} p_i > N$. Figure 3 illustrates the exponential decreasing of the error probability of the progressive verification procedure for 2048-bit RSA modulus (*i.e.* $\lambda = 2048$) and 12-bits prime numbers (*i.e.* $\mu = 12$).

### *3.4. Complexity, Implementation and Performances*

Using naive modular arithmetic, namely multiplication and division modulo $N$ with quadratic binary complexity (with no special tricks like Fast Fourier Transform), the binary complexity of the RSA-FDH.Verify algorithm is $O(\log(e) \cdot \lambda^2)$. The algorithm Progressive-RSA($\mu$).ProgVerify has improved binary complexity $O(\log(e) \cdot \lambda \cdot \mu)$ (since modular reduction of an $n$-bit integer by an $m$-bit integer requires $O(mn)$ binary operations, see [12]).

The proposed variant of the RSA signature scheme was implemented in order to validate its efficiency in practice. The implementation was done in C language using the GMP BIGNUM Library [30]. The scheme was tested with the public key $e = 2^{16} + 1$, which is a common choice in practice, and a good candidate to evaluate the performance of the progressive verification, with respect to the increase in the signature size. The results for RSA-2048 and RSA-4096 are given in Table 1. The desired upper bound on the error probability for the modified scheme has been fixed to $2^{-60}$, and different number of generated primes with different sizes have been tested accordingly to illustrate the trade-off discussed in 3.2. The verification modulo each prime was done independently in parallel. The progressive verification scheme shows faster execution when choosing higher number of primes, with smaller sizes. This behavior is normal since the reduction operation is less costly when the size of the prime decreases. In fact, the fastest verification is achieved when choosing four primes of 26 bits in the case of the modified RSA-2048, and four primes of 27 bits in the case of RSA-4096. It can also be observed that the verification process of the modified RSA scheme is clearly faster than the classical one. The verification time for either a valid or an invalid signature is the same with the latter for both RSA-2048 and RSA-4096, since the entire exponentiation of $\sigma^e \mod N$ have to be executed in order to validate the signature. Meanwhile, the verification for the valid signature is twice (resp. four times) faster in the case of progressive RSA-2048 (resp. RSA-4096). An invalid signature was constructed by injecting one single incorrect equation in a valid signature, and then randomly choosing the order of the equations to verify. An invalid signature is clearly detected faster with the progressive verification. On average, an invalid input is detected four times faster in the case of progressive RSA-2048 verification, and eight times faster for RSA-4096. This shows that the increase in the signature size leads to a much faster verification process for RSA signatures.

**Remark 2.** *In order to further improve the efficiency of our scheme, we considered to choose the prime numbers in the proposed RSA signature scheme, to be so-called* pseudo-Mersenne primes *[48], which offer faster modular reductions. However, this did not lead to an interesting improvement in the efficiency/security trade-off (since these special prime numbers are more rare and the error probability is significantly higher).*

| | Signature Size (in kB) | Verification Time (in ms) Valid | Invalid | (Number, Size) of primes |
|---|---|---|---|---|
| **RSA-2048** | 0.256 | 0.0291 | | Verification modulo $N$ ($\|N\| = 2048$) |
| **Proposed RSA-2048** | 8.704 | 0.0155 | 0.0073 | (1, 71) |
| | | 0.0139 | 0.0066 | (2, 41) |
| | | 0.0133 | 0.0064 | (3, 31) |
| | | **0.0130** | **0.0064** | **(4, 26)** |
| **RSA-4096** | 0.512 | 0.0873 | | Verification modulo $N$ ($\|N\| = 4096$) |
| **Proposed RSA-4096** | 17.408 | 0.0228 | 0.0087 | (1, 72) |
| | | 0.0182 | 0.0082 | (2, 42) |
| | | 0.0179 | 0.0081 | (3, 32) |
| | | **0.0177** | **0.0080** | **(4, 27)** |

Table 1: Performance of the progressive verification process, compared to the basic verification, in the cases of both valid and invalid signatures, for RSA-2048 and RSA-4096. The desired upper bound on the error probability has been fixed to $2^{-60}$.

## 4. ECDSA Signatures

### *4.1. Elliptic Curves and ECDSA Signatures*

The main advantage of elliptic curve cryptosystems stems from the absence of a sub-exponential-time algorithm to compute discrete logarithms on general elliptic curves over finite fields. Consequently, one can use an elliptic curve group that is smaller in size compared with systems in the multiplicative group of a finite field but the arithmetic of the underlying group is more tedious. Let $\mathbb{K}$ be a field. Consider an elliptic curve $E(\mathbb{K})$ defined over $\mathbb{K}$. $E(\mathbb{K})$ has the following generalized Weierstrass equation :

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{5}$$

where $a_1, a_2, a_3, a_4, a_5, a_6 \in \mathbb{K}$. For simplicity of notations and description of the ECDSA signature and the proposed fast verification scheme, we will consider[8] $\mathbb{K} = \mathbb{Z}_p$ of characteristic $p > 3$ a prime. Then, $E(\mathbb{Z}_p)$ admits a model using the short Weierstrass form :

$$y^2 = x^3 + ax + b \tag{6}$$

where $a, b \in \mathbb{Z}_p$ and $-16(4a^3 + 27b^2) \neq 0 \mod p$. Recently, a lot of work has been devoted to the study of alternative models of elliptic curves that could admit more efficient arithmetic (see [8] for an encyclopedic overview of these models). In this work, we consider only the (affine and projective Jacobian) Weierstrass model.

[8]Note that the proposed scheme would still apply for $\mathbb{K}$ of characteristic 2 or 3, changing only the form of the Weierstrass equation.

When speaking about point addition and doubling over elliptic curves, an estimate of their complexities will be discussed in number of basic field operations. In the next sections, the multiplication (resp. inversion) operation over the considered field $\mathbb{K}$, will be denoted by the symbol $\mathbf{M}$ (resp. $\mathbf{I}$).

We recall the description of the ECDSA signature scheme [38]. It achieves EUF-CMA-security in the generic group model and its security is related to the difficulty of the discrete logarithm problem. Our description involves a security parameter $\lambda \in \mathbb{N}$ (following Definition 1); this integer $\lambda$ represents the bit-size of the underlying finite field but not actually the security level of the signature scheme. To achieve a security level of $\kappa$ security bits, we must choose $\lambda \approx 2\kappa$ to prevent the best known attacks. The ECDSA signature scheme consists of the following procedures:

- $\mathsf{KeyGen}(1^\lambda)$: Generate a prime $p$ of bit-length $\lambda$ uniformly at random. Define an elliptic curve $E(\mathbb{Z}_p)$, along with a point $\boldsymbol{G} \in E$, of prime order $n$ (with $n$ also of bit-length $\lambda$). Then, choose $d \in \mathbb{Z}_n$ uniformly at random, and calculate the point $\boldsymbol{Q} = d\boldsymbol{G}$. The verification key is $\mathsf{vk} = (p, E(\mathbb{Z}_p), \boldsymbol{G}, \boldsymbol{Q}, \mathcal{H})$ where $\mathcal{H}$ is a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$, while the signing key is $\mathsf{sk} = (p, E(\mathbb{Z}_p), \boldsymbol{G}, d, \mathcal{H})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: The signer chooses a secret $k \in \mathbb{Z}_n$ uniformly at random, then calculates the point $(x_1, y_1) = k\boldsymbol{G}$, and $r = x_1 \mod n$. He repeats the whole process until $r \neq 0$. Next, the signer computes $s = k^{-1}(\mathcal{H}(m) + rd) \mod n$ and repeats the signing process from the beginning until $s \neq 0$. The signature of the message $m$ is the pair $\sigma = (r, s)$.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: To verify a signature pair $\sigma = (r, s)$ of a message $m$ for $\mathsf{vk}$, the verifier first checks that $\boldsymbol{Q} \neq \boldsymbol{O}$, and $0 < r, s < n$. He then computes the point $\boldsymbol{R} = (x, y) = u_1\boldsymbol{G} + u_2\boldsymbol{Q}$, where $u_1 = \mathcal{H}(m)s^{-1} \mod n$ and $u_2 = rs^{-1} \mod n$. He accepts the signature if and only if $(x, y) \neq \boldsymbol{O}$ and $r = x \mod n$.

*4.2. ECDSA Variant with Fast Verification*

In this section, we propose a variant of ECDSA signatures that produces longer signatures but allows faster deterministic verification. The idea behind our proposed verification scheme of ECDSA is to speedup the double exponentiation algorithm of $u_1\boldsymbol{G}+u_2\boldsymbol{Q}$, by using the point doubling and adding algorithms with classical affine coordinates, while delegating all the inverse computations to the signer, and verifying their values during the process.

*4.2.1. Description of the Scheme*

We consider the group law defined on the affine Weierstrass model (6). In this case, the double of a point $\boldsymbol{P} = (x_P, y_P)$ is the point $2\boldsymbol{P} = (x_{2P}, y_{2P})$, where:

$$\lambda_0 = \frac{3x_P^2 + a}{2y_P}, \qquad x_{2P} = \lambda_0^2 - 2x_P, \qquad y_{2P} = (x_P - x_{2P})\lambda_0 - y_P$$

while the sum of two points $\boldsymbol{P} = (x_P, y_P)$ and $\boldsymbol{V} = (x_V, y_V)$ in $E(\mathbb{Z}_p)\backslash\{\boldsymbol{O}\}$ is $\boldsymbol{S} = \boldsymbol{P} + \boldsymbol{V} = (x_S, y_S)$, where :

$$\lambda_1 = \frac{y_V - y_P}{x_V - x_P}, \qquad x_S = \lambda_1^2 - x_V - x_P, \qquad y_S = (x_P - x_S)\lambda_1 - y_P$$

Thus, the doubling (resp. adding) operation costs $2\mathbf{S}+2\mathbf{M}+1\mathbf{I}$ (resp. $1\mathbf{S}+2\mathbf{M}+1\mathbf{I}$). A useful trick introduced in [18] can also be used when the point $\boldsymbol{T} = 2\boldsymbol{P}+\boldsymbol{V} = (x_T, y_T)$ needs to be computed, by writing $\boldsymbol{T} = \boldsymbol{P} + (\boldsymbol{P} + \boldsymbol{V}) = \boldsymbol{P} + \boldsymbol{S}$. Using the following identity :

$$\lambda_2 = \frac{y_S - y_P}{x_S - x_P} = \frac{((x_P - x_S)\lambda_1 - y_P) - y_P}{x_S - x_P} = \frac{2y_P}{x_P - x_S} - \lambda_1$$

we have :

$$x_T = \lambda_2^2 - x_P - x_S, \qquad y_T = (x_P - x_T)\lambda_2 - y_P$$

The authors from [18] suggest that the intermediate computation of $y_S$ can thus be omitted, leaving a total of $2\mathbf{S}+3\mathbf{M}+2\mathbf{I}$ for a double-and-add operation. During the double exponentiation algorithm, the simple add operation never occurs. There is one double and one or no add at each iteration.

Computing $\lambda_0$, $\lambda_1$ or $\lambda_2$ involves an inverse computation in $\mathbb{Z}_p$ which is usually too costly to perform for fast signature verification. For this reason, alternative models are often preferable to the affine Weierstrass model. In our scheme, we deport the computation of these inverses to the signer that would append the values of $\lambda_0$, $\lambda_1$ and $\lambda_2$ when needed for each step of the "double-and-add" exponentiation algorithm. There are $\lambda$ iterations in this algorithm: if the iteration includes a simple double, the signer would join the associated value of $\lambda_0$ to the signature; if it includes a double-and-add operation, the signer would associate the values of $\lambda_1$ and $\lambda_2$.

During the verification process, each inversion is then replaced by a value check, adding no extra operations compared to the original scheme, and speeding up the process but with an increase in the signature size. For example, if the operation is a point addition, the signer would join the value of $\lambda_1$, and the verifier would check that

$$\lambda_1(x_V - x_P) = (y_V - y_P)$$

If this condition is verified, then the verifier can immediately use the value of $\lambda_1$. This means that the operations $1\mathbf{I}+1\mathbf{M}$ for computing $\lambda_1$ in the original scheme, are replaced by only $1\mathbf{M}$ for checking its value. Table 2 compares the operation count for doubling and adding points on elliptic curves over $\mathbb{Z}_p$, with both affine and Jacobian projective coordinates, using the best operation count for the short Weierstrass form. We assume that $\mathbf{S}$=$1\mathbf{M}$, and that the double-and-add operation with Jacobian projective coordinates is done by a simple double followed by a simple add. Compared to the operation count with projective coordinates, which are the coordinates usually used in practice to

| Type of Coordinates | Operation Count | | |
|---|---|---|---|
| | Double | Add | Double & Add |
| **Affine** | 4**M**+1**I** | 3**M**+1**I** | 5**M**+2**I** |
| **Affine with Provided Inverses** | 4**M** | 3**M** | 5**M** |
| **Projective Jacobian** | 9**M** | 16**M** | 25**M** |

Table 2: Operation count for double, add and double-and-add operations for elliptic curve of short Weierstrass form, assuming that **S**=1**M**.

avoid inversions, the proposed variant clearly offers a speedup of almost 2 for doubling a point, and of 5 for doubling-and-adding. Thus, it is more interesting during the double exponentiation algorithm to have as many iterations with double-and-add instead of a simple double as possible, to increase the speed of the computation. For that, we use another trick proposed in [2], that allows to reduce the number of iterations in half, while making the double-and-add operation more frequent during the execution. Consider the equation :

$$u_1\boldsymbol{G} + u_2\boldsymbol{Q} = \boldsymbol{R} \tag{7}$$

where $u_1$ and $u_2$ are of size $\lambda$. One can use the Euclidean algorithm as in [2], to write $u_2 = u_{21}u_{22}^{-1} \mod n$, with $|u_{21}|, |u_{22}| < \sqrt{n}$ (*i.e.* $u_{21}$ and $u_{22}$ are integers with bit-length at most $\lambda/2$). Equation (7) can then be rewritten :

$$u_1u_{22}\boldsymbol{G} + u_{21}\boldsymbol{Q} - u_{22}\boldsymbol{R} = \boldsymbol{O} \tag{8}$$

Then, one can write $u_1 = u_{11}2^{\lfloor\lambda/2\rfloor} + u_{12}$ with a simple Euclidean division of $u_1$ by $2^{\lfloor\lambda/2\rfloor}$. We can finally write (7) as :

$$\mu_0\boldsymbol{G} + \mu_1\boldsymbol{G_0} + u_{21}\boldsymbol{Q} - u_{22}\boldsymbol{R} = \boldsymbol{O} \tag{9}$$

where $\boldsymbol{G_0} = 2^{\lfloor\lambda/2\rfloor}\boldsymbol{G}$, $\mu_0 = u_{12}u_{22}$, $\mu_1 = u_{11}u_{22}$, and the bit-length of $\mu_0, \mu_1, u_{21}$ and $u_{22}$ is at most $\lambda/2$. This transforms the double exponentiation algorithm, into a quadruple exponentiation with only half the number of iterations of the double exponentiation. In addition, simple double operation will be much less frequent in the case of the quadruple exponentiation, making the speedup with our proposed method be more present with the double-and-add operation. In the best case scenario, where the number of simple double operations is negligible, one would expect a total speedup of factor almost 9 or 10 over the use of projective coordinates as shown in Table 2.

*4.2.2. Implementation and Performances*

To evaluate the performance of the proposed verification algorithm, the ECDSA signature scheme implemented in OpenSSL [51] was used. Some modifications were added to the key generation and signing processes, and a new verification process using the OpenSSL BIGNUM library was coded and compared to the classic double exponentiation verification algorithm, and the wNAF

| | Signature Size (in kb) | Verification Time (in ms) Valid | Invalid |
|---|---|---|---|
| **OpenSSL wNAF** | 0.064 | 1.81 | 1.81 |
| **Double exponentiation** | 0.064 | 2.65 | 2.65 |
| **Proposed variant** | 8.352 | 0.30 | 0.23 |

Table 3: Performance of the proposed progressive verification process, compared to the basic verification, in the cases of both valid and invalid signatures, for ECDSA-256 in OpenSSL, for the curve $E : y^2 = x^3 + 7$.

algorithm used by OpenSSL. Table 3 compares the execution time for the different procedures. It is clear that the proposed variant, while considerably increasing the size of the signature, offers a speedup in practice of almost 9 over the regular double exponentiation algorithm. It is also more efficient compared to the optimized version of the OpenSSL wNAF implementation, where a factor 6 of speedup is obtained. This shows a trade-off that can be offered, when the verification time is more critical than the signature size for the considered application.

**Remark 3.** *The new scheme is less efficient than the EdDSA variant from [7] (which produces also shorter signatures) but it is generic and works over any prime finite field and we did not try to propose an optimized implementation.*

**Remark 4.** *A verification time / signature length trade-off can be offered with the compression technique introduced in [13]. For a double-and-add operation, the values $\lambda_1$ and $\lambda_2$ can be compressed and added to the signature in a single value. Meanwhile, the number of multiplications required for the operation becomes roughly $11\boldsymbol{M}$. This means that the trade-off reduces the signature size by half, while allowing a decrease in the speedup factor from 5 to approximately 2.3.*

### *4.3. ECDSA Variant with Progressive Verification*

Our trick of using affine coordinates and giving the inverses in the elliptic curve group law gives rise to a more efficient "progressive-verification" scheme for ECDSA. It uses the same technique than for RSA: for each arithmetic operation modulo $p$ or modulo the group order, one provides the quotient of the Euclidean division and we check the equality of integers modulo some "small" prime number of bit-length $\mu$. This increases even more the length of the signature but makes the verification algorithm progressive and faster. These digital signatures with progressive verification called Progressive-ECDSA($\mu$) are very similar to the Progressive-RSA signature scheme. The technique consists in checking all $O(\lambda)$ equalities modulo $\tau$ distinct random prime numbers of bit-length $\mu$; if one equality does not hold, the algorithm ProgVerify outputs $\perp$ and if all equalities hold the $\tau$ prime numbers, it outputs the real $\alpha = 1 - (\lambda \cdot 2^{-\mu})^{\tau}$. We leave the

details to the reader. We obtain readily the following security results (whose proofs follow *mutatis mutandis* the proofs from Proposition 1 and Theorem 1, respectively).

**Proposition 2.** *Let $t : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to [0,1]$ and $\mathcal{A}$ be a $(t,\epsilon)$-*EUF-CMA*-adversary against* Progressive-ECDSA. *There exist $\mathcal{B}$ a $(t',\epsilon)$-*EUF-CMA*-adversary against* ECDSA *with*

$$t'(\lambda) = t(\lambda) + q_S O(\lambda^3)$$

*and $q_S$ denotes the number of queries made by $\mathcal{A}$ to its signing oracle.*

**Theorem 2.** *Let $\mu \in \mathbb{N}$ with $\mu > 5$. The digital signature scheme with progressive verification* Progressive-ECDSA$(\mu)$ *is* correct.

Progressive-ECDSA$(\mu)$ produces signature size with size $O(\lambda^2) = O(\kappa^2)$ (which are much longer than classical ECDSA signatures but still asymptotically shorter than RSA signatures that has bit-length $\Omega(\kappa^3)$ for security level $\kappa$). The (deterministic) fast verification algorithm has complexity $O(\lambda^3)$ binary operations (similar to ECDSA signatures but with a better constant) and the progressive verification algorithm has complexity $O(\tau \cdot \lambda^2 \cdot \mu)$ binary operations (where the error probability of the verifier decreases exponentially with $\tau$).

## 5. GPV Signatures and Variants

### *5.1. Lattices and GPV Signatures*

We first start by providing a brief background on lattices that will be useful for the description of the GPV signature scheme. All details are not necessary to understand our scheme with progressive verification but we provide them for completeness. Let $p \geq 2$ be some prime integer and let $\boldsymbol{A} \in \mathbb{Z}_p^{n \times m}$ be a matrix formed of $n$ linearly independent vectors denoted $\boldsymbol{A} = \{\vec{a_1}, \ldots, \vec{a_n}\}$ with $\vec{a_i} \in \mathbb{Z}_p^m$. The $m$-dimensional lattice associated to $\boldsymbol{A}$ is:

$$\Lambda(\boldsymbol{A}) = \{\vec{x} \in \mathbb{Z}_p^m : \boldsymbol{A}^T s = \vec{x} \bmod p \text{ for some } s \in \mathbb{Z}_p^n\}$$

and its dual $\Lambda^{\perp}(\boldsymbol{A}) = \{\vec{e} \in \mathbb{Z}_p^m : \boldsymbol{A}\vec{e} = 0 \bmod p\}$.

One usually considers $n$ as the security parameter and $p$ and $m$ as functions of $n$, that is $p = \mathsf{poly}(n)$, $m = \mathsf{poly}(n)$. For any $\vec{c} \in \mathbb{R}^n$, $s \in \mathbb{R}$, we define the discrete Gaussian distribution over $\Lambda(\boldsymbol{A})$ as:

$$D_{\Lambda(\boldsymbol{A}),s,\vec{c}}(\vec{x}) = \frac{\rho_{s,\vec{c}}(\vec{x})}{\rho_{s,\vec{c}}(\Lambda(\boldsymbol{A}))}, \text{ for all } \vec{x} \in \Lambda(\boldsymbol{A})$$

where $\rho_{s,\vec{c}}(\vec{x}) = \exp\left(-\pi \left\|\vec{x} - \vec{c}\right\|^2 / s^2\right)$ is the Gaussian function centered at $\vec{c}$ with parameter $s$ over $\mathbb{R}^n$ (and naturally $\rho_{s,\vec{c}}(\Lambda(\boldsymbol{A})) = \sum_{\vec{z} \in \Lambda(\boldsymbol{A})} \rho_{s,\vec{c}}(\vec{z})$). For a given basis $\boldsymbol{A} = \{\vec{a_1}, \ldots, \vec{a_n}\}$ of a lattice, the norm of the basis is defined as $\|\boldsymbol{A}\| = \max_{1 \leq i \leq n} \|\vec{a_i}\|$ where $\|\vec{a_i}\|$ is the Euclidean norm of $\vec{a_i}$.

A variety of hard problems have been studied on lattices, of which we focus on two, known as the Short Integer Solution (SIS) problem and the Inhomogeneous Short Integer Solution (ISIS) problem:

**Definition 4** ($SIS_{p,m,\beta}$)**.** *Given an integer $p$, a matrix $\boldsymbol{A} \in \mathbb{Z}_p^{n\times m}$, and a real $\beta$, the Short Integer Solution (SIS) problem with parameters $(p, m, \beta)$ is to find $\vec{e} \in \Lambda^{\perp}(\boldsymbol{A})$ such that $\|e\| \leq \beta$.*

**Definition 5** ($ISIS_{p,m,\beta}$)**.** *Given an integer $p$, a matrix $\boldsymbol{A} \in \mathbb{Z}_p^{n\times m}$, a vector $\vec{u} \in \mathbb{Z}_p^n$ and a real $\beta$, the Inhomogeneous Short Integer Solution (ISIS) problem with parameters $(p, m, \beta)$ is to find $\vec{e} \in \mathbb{Z}_p^m$ such that $\boldsymbol{A}\vec{e} = \vec{u} \bmod p$ and $\|e\| \leq \beta$.*

The GPV signature scheme [27] relies on the hardness of these problems on average, for chosen parameters (see [27, 3] for details). To construct the scheme, Gentry, Peikert and Vaikutanathan provided a tuple of of probabilistic polynomial-time algorithms (TrapGen, Sample, PreImageSample) that define a function $f_{\boldsymbol{A}} : D_n \to R_n$ (together with its domain $D_n$ and range $R_n$) associated to $\boldsymbol{A}$ with a "short" basis of $\Lambda^{\perp}(\boldsymbol{A})$ as a trapdoor for inverting $f_{\boldsymbol{A}}$. These algorithms are parameterized by a bound $L \approx m^{1+\epsilon}$ for $\epsilon > 0$, and a Gaussian parameter $s \geq L.\omega(\sqrt{\log m})$. In particular:

- $\mathsf{TrapGen}(1^n)$ : Use a probabilistic algorithm to sample a matrix $\boldsymbol{A} \in \mathbb{Z}_p^{n\times m}$ from a close-to-uniform distribution, and find a short basis $\boldsymbol{T} = \{\vec{t_1}, \ldots, \vec{t_n}\} \subset \Lambda^{\perp}(\boldsymbol{A})$ with $||\tilde{\boldsymbol{T}}|| \leq L$, where $\tilde{\boldsymbol{T}}$ is the Gram-Schmidt orthogonalization of $\boldsymbol{T}$ (see [1] for details). The function $f_{\boldsymbol{A}}(.)$ is then defined as $f_{\boldsymbol{A}}(\vec{e}) = \boldsymbol{A}\vec{e} \bmod p$ with $D_n = \{\vec{e} \in \mathbb{Z}_p^m : \|\vec{e}\| \leq s\sqrt{m}\}$, $R_n = \mathbb{Z}_p^n$ and $\boldsymbol{T}$ is its trapdoor.
- $\mathsf{Sample}(1^n)$ : Samples $\vec{e} \in \mathbb{Z}_p^m$ at random from the input distribution $D_{\mathbb{Z}_p^m, s, \vec{0}}$ conditioned by the output of $f_{\boldsymbol{A}}(\vec{e})$ for which the distribution is uniform over $R_n$.
- $\mathsf{PreImageSample}(\boldsymbol{T}, \vec{y})$ : Samples from $f_{\boldsymbol{A}}^{-1}(\vec{y})$ using the "good" trapdoor basis $\boldsymbol{T}$ to output $\vec{e} \in D_n$, which means that $\vec{e}$ is a short vector.

Without the trapdoor, sampling a short element from $D_n$ is hard. It is proven in [27] that these algorithms provide a one-way trapdoor function if $ISIS_{p,m,s\sqrt{m}}$ is hard. They are also collision resistant if $SIS_{p,m,2s\sqrt{m}}$ is hard.

- $\mathsf{KeyGen}(1^\lambda)$: Given $\lambda$, it sets the parameter $n = \lambda$ and $m$, $p$, $s$, $D_n$, $R_n$ and $f_{\boldsymbol{A}}$ as defined above for a pair $(\boldsymbol{A}, \boldsymbol{T})$ where $\boldsymbol{A} \in \mathbb{Z}_p^{n\times m}$ and $\boldsymbol{T}$ is the trapdoor, from $\mathsf{TrapGen}(1^n)$. The verification key is $\mathsf{vk} = (\boldsymbol{A}, \mathcal{H})$ where $\mathcal{H}$ is a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p^n$ , while the signing key is $\mathsf{sk} = (\boldsymbol{T}, \mathcal{H})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: Compute $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$, and run $\mathsf{PreImageSample}(\boldsymbol{T}, \vec{y})$ to get $\vec{u} \in D_n$. Output the signature $\sigma = \vec{u}$.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Compute $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$, and return 1 if $\boldsymbol{A}\vec{u} = \vec{y} \bmod p$ and $\vec{u} \in D_n$ (in other words $\|u\| \leq s\sqrt{m}$) and 0 otherwise.

It is worth noting that since $\mathcal{H}$ is a hash function which output is uniformly distributed over $\mathbb{Z}_p^n$. This ensures the condition stated in the procedure $\mathsf{Sample}(1^n)$, where the output of $f_{\boldsymbol{A}}(e)$ should be uniformly distributed over $R_n = \mathbb{Z}_p^n$.

*5.2. GPV with Progressive Verification*

As outlined in the paper introduction, our progressive variant of the GPV signature scheme denoted Progressive-GPV introduces the use of linear codes over $\mathbb{Z}_p$.

For this reason, we consider in the following that $p$ is a prime number. Let $\ell, n \in \mathbb{N}$ and $\delta \in [0,1]$. An $(\ell, n, \delta\ell)$ $p$-ary linear code $C$ is a $n$-dimensional linear subspace of $\mathbb{Z}_p^\ell$ in which the Hamming distance between each two distinct vectors (called codewords) is at least $\delta\ell$ ($\delta$ is called the *relative distance* of the code). For our purpose[9], a matrix $\boldsymbol{G}$ in $\mathbb{Z}_p^{\ell \times n}$ is a *generator matrix* of $C$ if its columns space over $\mathbb{Z}_p$ (*i.e.* the set of $\mathbb{Z}_p$-linear combinations of its columns) is equal to $C$. The ratio $R = n/\ell$ is called the *rate* of $C$.

The security analysis of the scheme will be detailed in Section 5.3. The progressive variant Progressive-GPV of the GPV signature scheme consists of the following three algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: Given $\lambda$, it sets the parameter $n = \lambda$ and $m$, $p$, $s$, $D_n$, $R_n$ and $f_{\boldsymbol{A}}$ as defined above for a pair $(\boldsymbol{A}, \boldsymbol{T})$ where $\boldsymbol{A} \in \mathbb{Z}_p^{n \times m}$ and $\boldsymbol{T}$ is the trapdoor, from $\mathsf{TrapGen}(1^n)$. The algorithm also sets two parameters $\ell \in \mathbb{N}$ with $\ell > n$ and $\delta \in [0,1]$ and generate a $(\ell, n, \delta\ell)$ linear code with generator matrix $\boldsymbol{G} \in \mathbb{Z}_p^{\ell \times n}$. It sets $\boldsymbol{A'} = \boldsymbol{G} \cdot \boldsymbol{A}$. The verification key is $\mathsf{vk} = (\boldsymbol{A'}, \boldsymbol{G}, \mathcal{H})$ where $\mathcal{H}$ is a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p^n$, while the signing key is $\mathsf{sk} = (\boldsymbol{T}, \mathcal{H})$.

- $\mathsf{Sign}(\mathsf{sk}, m)$: Compute $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$, and run $\mathsf{PreImageSample}(\boldsymbol{T}, \vec{y})$ to get $\vec{u} \in D_n$. Output the signature $\sigma = \vec{u}$.

- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Compute $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$ and set $\vec{u} = \sigma$. Return 1 if $\boldsymbol{A'}\vec{u} = \boldsymbol{G}\vec{y} \bmod p$ and $\vec{u} \in D_n$ (in other words $\|u\| \leq s\sqrt{m}$) and 0 otherwise.

In this scheme, the signing algorithm and the signatures are not modified (compared to the original GPV scheme). The size of the public key is increased (from $nm\log(p)$ to $\ell(n+m)\log(p)$ bits) and the verification algorithm is slower than the verification algorithm of the GPV scheme (since it requires two matrix/vector multiplication instead of just one). However, the use of a "good" error-correcting code of generating matrix $\boldsymbol{G}$ allows fast progressive verification since a verifier can simply check the equality of random rows of $\boldsymbol{A'}\vec{u}$ and $\boldsymbol{G}\vec{y}$ with $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$ for a putative signature $\sigma = \vec{u}$ of a message $m$:

- $\mathsf{ProgVerify}(\mathsf{vk}, m, \sigma, \tau)$: Given a putative signature $\sigma$ of a message $m$ for $\mathsf{vk}$, compute $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$, set $\vec{u} = \sigma$ and check that $\vec{u} \in D_n$ (i.e $\|u\| \leq s\sqrt{m}$). The algorithm picks uniformly at random $\tau$ distinct indices $j_1, \ldots, j_\tau$ in $[\ell]$ and returns $\perp$ if $\boldsymbol{A'}\vec{u}\{j_i\} \neq \boldsymbol{G}\vec{y}\{j_i\} \bmod p$ for some $i \in [\tau]$ and it returns $\alpha = 1 - (1-\delta)^\tau$ otherwise.

[9] Note that usually one considers a generator matrix as a matrix whose rows form a basis of the code, but using the columns makes the presentation simpler.

In this algorithm, for a vector $\vec{v} \in \mathbb{Z}_p^\ell$ and $i \in [\ell]$, $\vec{v}\{i\}$ denotes the $i$-th coordinate of $\vec{v}$. In particular, for $i \in [\tau]$ computing $\boldsymbol{A'}\vec{u}\{j_i\}$ is an inner-product over $\mathbb{Z}_p$ of vectors of length $n$ and computing $\boldsymbol{G}\vec{y}\{j_i\}$ is an inner-product over $\mathbb{Z}_p$ of vectors of length $m$. Note that the larger $\delta$ is, the more accurate the progressive verification procedure, but the larger the public-key size .

*5.3. Security Analysis*

The signatures output by Progressive-GPV are identical to those produces by GPV. Moreover, the algorithm Progressive-GPV.Verify accepts $\sigma$ as a valid signature on $m$ for vk if and only if we have GPV.Verify$(\mathsf{vk}, m, \sigma) = 1$ (since $\boldsymbol{G} \in \mathbb{Z}_p^{\ell \times n}$ is of full-rank $n$). We thus obtain immediately the following proposition:

**Proposition 3.** *Let $t : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to [0, 1]$, be two functions and let $\mathcal{A}$ be $(t, \epsilon)$-*EUF-CMA*-adversary against Progressive-GPV. There exist $\mathcal{B}$ a $(t', \epsilon)$-*EUF-CMA*-adversary against GPV with $t' = t$.*

□

Following the probabilistic analysis outlined in the introduction, we can show that the progressive verification of Progressive-GPV is correct:

**Theorem 3.** *The signature scheme with progressive verification Progressive-GPV is* correct *(*i.e. *it returns $\perp$ only for invalid signatures and if it returns a real number $\alpha \in [0, 1]$ on input $(\mathsf{vk}, m, \sigma, \tau)$ then $\sigma$ is not a signature of $m$ for vk with probability at most $1 - \alpha$.*

*Proof.* Given a putative signature $\sigma$ of a message $m$ for vk, $\sigma = \vec{u}$ is valid if and only $\boldsymbol{A'}\vec{u} = \boldsymbol{G}\vec{y} \bmod p$ where $\vec{y} = \mathcal{H}(m) \in \mathbb{Z}_p^n$. In other words, $\sigma$ is valid if and only if

$$\boldsymbol{G}(\boldsymbol{A}\vec{u} - \vec{y}) = \vec{0} \bmod p.$$

If $\sigma$ is valid the all $\tau$ random rows checked during the run of ProgVerify$(\mathsf{vk}, m, \sigma, \tau)$ are equal and ProgVerify$(\mathsf{vk}, m, \sigma, \tau)$ does not return $\perp$. Besides, if $\sigma$ is not valid, at least one row out of the $\ell$ are not equal and we have $\boldsymbol{G}(\boldsymbol{A}\vec{u} - \vec{y}) \neq \vec{0} \bmod p$. By construction, $\boldsymbol{G}(\boldsymbol{A}\vec{u} - \vec{y})$ is a codeword in our error-correcting code of generator matrix $\boldsymbol{G}$ which is a $(\ell, n, \delta\ell)$ linear code. A non-zero codeword has Hamming weight (*i.e.* non-zero coordinates) at least $\delta \cdot \ell$ and thus there exists at least $\delta\ell$ coordinates $k_1, \ldots, k_\omega$ in $[\ell]$ for $\omega \geq \delta\ell$ such that $\boldsymbol{A'}\vec{u}\{k_i\} \neq \boldsymbol{G}\vec{y}\{k_i\} \bmod p$ for $i \in [\omega]$. The probability that each row index $j_i$ picked in ProgVerify does not belong to $\{k_1, \ldots, k_\omega\}$ is upper-bounded by $(\ell - \omega)/\ell \leq (\ell - \delta\ell)/\ell = (1 - \delta)$ and therefore the probability that the $\tau$ row indices picked by ProgVerify does not belong to this set is upper-bounded by $(1 - \delta)^\tau$. The probability that an invalid signature is not detected by the verification algorithm is upper-bounded by $(1 - \delta)^\tau$ and thus the real $\alpha = 1 - (1 - \delta)^\tau$ output by the algorithm provides a correct confidence level for this signature scheme. □

*5.4. Complexity, Implementation and Performances*

The proposed scheme Progressive-GPV produces signature identical to those produced by the original GPV (with the same algorithm and the same computational complexity). The verification key in Progressive-GPV is larger since it consists of two-matrices over $\mathbb{Z}_p$ instead of just one: the verification key size is $\ell(n+m)\log(p)$ instead of $nm\log(p)$ (with $\ell > n$). For the progressive verification algorithm, the error probability decreases exponentially with $\tau$ the number of verifier steps as $(1-\delta)^\tau$. Its binary computational complexity of the algorithm is $O(\tau(n+m)\log^2(p))$ (compared to $O(nm\log^2(p))$ for the original GPV scheme) and does not depend on $\ell$. It is therefore desirable to have an error-correcting code with $\ell$ as small as possible and $\delta$ as large as possible. Since we do not need any other specific property for our error-correcting (such as efficient decoding), we simply use random linear error-correcting codes which achieve the so-called *Gilbert-Varshamov bound* with overwhelming probability.

*5.4.1. Random linear codes and Gilbert-Varshamov Bound*

The Gilbert-Varshamov bound for linear codes asserts the existence of $p$-ary linear codes for any relative minimum distance less than some given bound that simultaneously have high rate. Let $H_p$ denote the $p$-ary entropy function:

$$H_p(x) = x\log_p(p-1) - x\log_p x - (1-x)\log_p(1-x).$$

We have the following result:

**Theorem 4** ([52, 28]). *Let $p \geqslant 2$ be some prime number. For every $0 \leqslant \delta < 1 - p^{-1}$ and $0 < \varepsilon \leqslant 1 - H_p(\delta)$, there exists a linear code with rate $R \geqslant 1 - H_p(\delta) - \varepsilon$ and relative distance at least $\delta$.*

It is worth mentioning that the existence proof of this theorem uses the probabilistic method, and thus is not constructive. However, it can be made explicit in the sense that the probability that a random code over $\mathbb{Z}_p$ with a given rate does not achieve the relative distance given by Theorem 4 is negligible:

**Corollary 1** ([52]). *Let $p \geqslant 2$ be some prime number. Let $n, \ell$ be two integers with $\ell > n$ and let $\delta \in [0,1]$. The probability that a code defined by a generator matrix $\boldsymbol{G}$ picked uniformly at random in $\mathbb{Z}_p^{\ell\times n}$ has relative distance smaller than $\delta$ is smaller than $p^{-\ell(1-H_p(\delta))+n}$.*

*5.4.2. Execution Time for Progressive-GPV*

In this paragraph, we consider implementation for GPV signatures for two parameters settings for GPV signatures taken from [3] (but using a prime value for $p$ with similar size to the power of 2 considered in [3]). The first instantiation provides a 78-bit security level (with a prime $p$ with bit-length 27) while the second one provides a 128-bit security level (with a prime $p$ with bit-length 30).

Suppose we want an error-correcting code with rate $R = 1/2$. We use a random generator matrix $\boldsymbol{G} \in \mathbb{Z}_p^{\ell\times n}$ with $\ell = 2n$ (where $n = 256$ for the first instantiation and $n = 512$ for the second instantiation). For $\delta = 0.45$, we have

| GPV Configuration | Security Level | Basic Verification Time (in ms) | Code Rate | $\delta$ | Progressive Verification Time (in ms) |
|---|---|---|---|---|---|
| $p \approx 2^{27}$, $n = 256$, $m = 7424$ | 78 bits | 24.76 | 1/2 | 0.45 | 12.505 |
| | | | 1/3 | 0.62 | 8.410 |
| | | | 1/4 | 0.70 | 6.134 |
| $p \approx 2^{30}$, $n = 512$, $m = 16384$ | 108 bits | 83.203 | 1/2 | 0.45 | 19.054 |
| | | | 1/3 | 0.62 | 13.313 |
| | | | 1/4 | 0.70 | 9.898 |

Table 4: Progressive Verification configuration and execution time on GPV signature Scheme, using a randomly generated $(\ell, n, \delta\ell)$ linear code. The error bound for the verification is fixed at $2^{-96}$. The verification is simulated using the MPFQ Finite Fields Library [26].

in both cases $H_p(\delta) < 0.487$ and thus the probability given by Corollary 1 is upper-bounded by $p^{-\ell(1-H_p(\delta))+n} = p^{-n(1-2H_p(\delta))} \leq p^{-0.026n} < 2^{-179}$. We can therefore safely assume that our random matrix generates a code with relative distance at least $\delta = 0.45$ even if there is no efficient way to check that this is indeed the case. Formally, in our security analysis, we have to add this small probability to the error probability in the progressive verification decision procedure. We also consider smaller rates $R = 1/3$ and $R = 1/4$ that increase the size of the verification key but ensures that we have a code with the large relative distance with overwhelming probability (namely $\delta \geq 0.24$ and $\delta \geq 0.29$ respectively). As proved above, the error probability of the progressive verification algorithm decreases as $\delta^\tau$ where $\tau$ is the number of random rows tested in the procedure. For the first case with rate $R = 1/2$ and $p$ a prime number of bit-length 27, the time to check one row with our parameter is approximatively 0.112 ms using the MPFQ Finite Fields Library [26]. By checking 38 rows in time 4.26 ms, we have an error probability at most $2^{-32}$ and if one checks 112 rows (which makes the error probability as small as $2^{-96}$), the time verification is only 12.505 ms (compared to to 24.76 ms for the original verification of GPV for these parameters). Table 4 summarized our obtained results for the two instantiations and rates $R \in \{1/4, 1/3, 1/2\}$.

**Remark 5.** *A possible approach to limit the public-key size increase in our scheme is to use a matrix $\boldsymbol{G}$ with some structure in order to compress it. For instance, one can use cyclic random linear codes or quasi-cyclic random linear codes. Bounds similar to the Gilbert-Varshamov bound are known for such codes (see [25] for instance) for p large enough (and with some specific arithmetical properties). This allows to compress the verification key (but at the cost of reducing the proven relative distance and thus the verification efficiency).*

*5.5. Wave Signatures and Progressive Verification*

*5.5.1. Brief description*

Recently, Debris-Alazard, Sendrier and Tillich [16] presented an efficient code-based signature scheme called Wave. We do not describe their nice scheme in details but rather refer to [16] for details. As GPV signatures, Wave is based on a family of trapdoor one-way preimage sampleable functions $f$; the message to be signed is hashed to produce a random element in the domain of $f$ and the signer uses the trapdoor to compute a pre-image of this element. Debris-Alazard *et al.* proposed a code-based one way trapdoor function that meets the preimage sampleable property (in a slighty relaxed way). Instead of a short basis of the lattice considered in the construction as in GPV, their trapdoor consists in choosing parity-check matrices of specific error-correcting codes over the ternary field $\mathbb{F}_3$. For our purposes, it is sufficient to know that the verification algorithm of Wave consists of a single matrix/vector multiplication and a check that the result matches the hash value of the message (and some given random salt). The underlying finite field in $\mathbb{Z}_3$ and we can use our progressive verification technique even on this small finite field (but a smaller relative distance).

*5.5.2. Execution Time for Progressive-Wave*

Debris-Alazard *et al.* proposed several parameters and an efficient implementation and we include our progressive verification algorithm to their implementation. For a 96-bit security level, using the same notation as above the matrix used in the (deterministic) verification algorithm is $n \times m$ with $n = 2165$ and $m = 4203$. For a rate $R = 1/2$, since $p = 3$, we cannot obtain a very large relative distance for our random linear code, but with $\delta = 0.15$, we have $H_3(\delta) \leq 0.48$ and the probability given by Corollary 1 is upper-bounded by $p^{-\ell(1-H_p(\delta))+n} = 3^{-n(1-2H_p(\delta))} \leq 3^{-0.04n} < 2^{-137}$. We can therefore again safely assume that our random matrix generates a code with relative distance at least $\delta = 0.15$. Table 5 shows the execution time of the basic and progressive verification schemes. As for GPV, we present timings for a full progressive verification that makes the error probability smaller than $2^{-96}$.

## 6. Conclusion and Open Problems

We revisited the concepts of progressive verification and flexible signatures. We proposed efficient probabilistic verification procedures for RSA, ECDSA, GPV and Wave digital signatures in which the error probability of a verifier decreases exponentially with the verifier running time.

Our proposed scheme for ECDSA increases significantly the signature size; an interesting open problem is to propose a scheme with efficient progressive verification where this size is only increased by some constant factor (or prove that none exists). It also remains open to design such verification procedures for pairing-based signature schemes such as [10] or [9]. It may be possible to adapt the techniques proposed in this article for pairing computation, for instance by

| Wave Configuration | Security Level | Basic Verification Time (in ms) | Code Rate | $\delta$ | Progressive Verification Time (in ms) |
|---|---|---|---|---|---|
| $q = 3$, $n = 2165$, $m = 4203$ | 96 bits | 1.416 | 1/2 | 0.15 | 0.424 |
| | | | 1/3 | 0.24 | 0.291 |
| | | | 1/4 | 0.29 | 0.223 |
| $q = 3$, $n = 2887$, $m = 5605$ | 128 bits | 2.614 | 1/2 | 0.15 | 0.615 |
| | | | 1/3 | 0.24 | 0.375 |
| | | | 1/4 | 0.29 | 0.294 |

Table 5: Progressive Verification configuration and execution time for the Wave signature Scheme, using a randomly generated $(\ell, n, \delta\ell)$ linear code. The error bound for the verification is fixed at $2^{-96}$.

appending to the signatures the coefficients arising in the lines and tangent computation of the Miller functions (as it has been done for pairing outsourcing for optimal Ate pairing on a Barreto-Naehrig curve in [31]). However this requires a specific efficiency analysis and new ideas are probably required to keep the signature length practical. Finally, since verification procedures are involved in most cryptographic primitives, it is interesting to see if our techniques permit improvement in other settings (*e.g.* for commitment schemes, or proof systems).

## References


[1] Miklós Ajtai. Generating hard instances of the short basis problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1–9. Springer, 1999.

[2] Adrian Antipa, Daniel R. L. Brown, Robert Gallant, Rob Lambert, René Struik, and Scott A. Vanstone. Accelerated verification of ECDSA signatures. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 307–318. Springer, Heidelberg, August 2006.

[3] Rachid El Bansarkhani and Johannes Buchmann. Improvement and efficient implementation of a lattice-based signature scheme. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 48–67. Springer, Heidelberg, August 2014.

[4] Mihir Bellare, Juan A. Garay, and Tal Rabin. Batch verification with applications to cryptography and checking. In Claudio L. Lucchesi and Arnaldo V. Moura, editors, *LATIN 1998*, volume 1380 of *LNCS*, pages 170–191. Springer, Heidelberg, April 1998.

[5] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Heidelberg, May 1996.

[6] Daniel J. Bernstein. A secure public-key signature system with extremely fast verification. unpublished, available at `https://cr.yp.to/papers/sigs.pdf`, 2000.

[7] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89, 2012.

[8] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. `http://www.hyperelliptic.org/EFD`, download date: 2019-12-10.

[9] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[10] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[11] Cecilia Boschini, Dario Fiore, and Elena Pagnin. On the efficiency and flexibility of signature verifications formal model and realizations from lattices. (in submission), 2020.

[12] Richard Brent and Paul Zimmermann. *Modern Computer Arithmetic*. Cambridge University Press, New York, NY, USA, 2010.

[13] Mathieu Ciet, Marc Joye, Kristin Lauter, and Peter L Montgomery. Trading inversions for multiplications in elliptic curve cryptography. *Designs, codes and cryptography*, 39(2):189–206, 2006.

[14] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000.

[15] Ch. J. de la Vallée Poussin. Recherches analytiques sur la théorie des nombres premiers. Brux. S. sc. 21 B, 183-256, 281-362, 363-397, 1896.

[16] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings*, volume to appear of *Lecture Notes in Computer Science*. Springer, 2019.

[17] Pierre Dusart. *Autour de la fonction qui compte le nombre de nombres premiers.* PhD thesis, Université de Limoges, 1998.

[18] Kirsten Eisenträger, Kristin Lauter, and Peter L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 343–354. Springer, Heidelberg, April 2003.

[19] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, June 1988.

[20] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[21] Marc Fischlin. Progressive verification: The case of message authentication: (extended abstract). In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 416–429. Springer, Heidelberg, December 2003.

[22] Marc Fischlin. Fast verification of hash chains. In Tatsuaki Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 339–352. Springer, Heidelberg, February 2004.

[23] Cody Freitag, Rishab Goyal, Susan Hohenberger, Venkata Koppula, Eysa Lee, Tatsuaki Okamoto, Jordan Tran, and Brent Waters. Signature schemes with randomized verification. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 373–389. Springer, Heidelberg, July 2017.

[24] Rusins Freivalds. Probabilistic machines can use less running time. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, pages 839–842. North-Holland, 1977.

[25] Philippe Gaborit and Gilles Zémor. Asymptotic improvement of the Gilbert-Varshamov bound for linear codes. *IEEE Trans. Information Theory*, 54(9):3865–3872, 2008.

[26] Pierrick Gaudry and Emmanuel Thomé. The MPFQ library and implementing curve-based key exchanges, 2007.

[27] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[28] E. N. Gilbert. A comparison of signalling alphabets. *The Bell System Technical Journal*, 31(3):504–522, 1952.

[29] Marc Girault and David Lefranc. Server-aided verification: Theory and practice. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 605–623. Springer, Heidelberg, December 2005.

[30] Torbjörn Granlund and et al. GNU multiple precision arithmetic library 4.1.2, December 2002. `http://swox.com/gmp/`.

[31] Aurore Guillevic and Damien Vergnaud. Algorithms for outsourcing pairing computation. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 193–211. Springer, 2014.

[32] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, Heidelberg, May 1988.

[33] J. Hadamard. Sur la distribution des zéros de la fonction $\zeta(s)$ et ses conséquences arithmétiques. *Bull. Soc. Math. Fr.*, 24:199–220, 1896.

[34] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 332–354. Springer, Heidelberg, August 2001.

[35] M. Jakobsson. Fractal hash sequence representation and traversal. In *Proceedings IEEE International Symposium on Information Theory,*, page 437, 2002.

[36] Markus Jakobsson. Fractal hash sequence representation and traversal. Cryptology ePrint Archive, Report 2002/001, 2002. `http://eprint.iacr.org/2002/001`.

[37] Markus Jakobsson, Frank Thomson Leighton, Silvio Micali, and Michael Szydlo. Fractal Merkle tree representation and traversal. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 314–326. Springer, Heidelberg, April 2003.

[38] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.*, 1(1):36–63, 2001.

[39] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 372–389. Springer, Heidelberg, December 2008.

[40] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

[41] Duc Viet Le, Mahimna Kelkar, and Aniket Kate. Flexible signatures: Making authentication suitable for real-time environments. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part I*, volume 11735 of *LNCS*, pages 173–193. Springer, Heidelberg, September 2019.

[42] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.

[43] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990.

[44] Albrecht Petzoldt, Ming-Shing Chen, Bo-Yin Yang, Chengdong Tao, and Jintai Ding. Design principles for HFEv-based multivariate signature schemes. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 311–334. Springer, Heidelberg, November / December 2015.

[45] Jean-Jacques Quisquater and Marijke De Soete. Speeding up smart card RSA computation with insecure coprosessors. In *Proceeding of Smart Cards 2000*, pages 191–197, 1989.

[46] Koichi Sakumoto, Taizo Shirai, and Harunaga Hiwatari. Public-key identification schemes based on multivariate quadratic polynomials. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 706–723. Springer, Heidelberg, August 2011.

[47] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[48] Jerome A. Solinas. Pseudo-Mersenne prime. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

[49] Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 13–21. Springer, Heidelberg, August 1994.

[50] Jacques Stern. A new paradigm for public key identification. *IEEE Trans. Information Theory*, 42(6):1757–1768, 1996.

[51] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. `www.openssl.org`, April 2003.

[52] R. R. Varshamov. Estimate of the number of signals in error correcting codes) (Russian, P. P. Варшамов – Оценка числа сигналов в кодах с коррекцией ошибок). *Dokl. Akad. Nauk SSSR*, 117:739–741, 1957.

[53] David Wagner. Shortcut digital signature verification failure. The Cryptography Mailing List, June, 22nd 2002.

[54] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 887–898. ACM Press, May 2012.