SIGNING ON A POSTCARD

David Naccache
Gemplus Card International
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
naccache@gemplus.com

Jacques Stern
Ecole Normale Supérieure
45 rue d'Ulm
Paris CEDEX 5, F-75230, France
jacques.stern@ens.fr

Abstract. We investigate the problem of signing short messages using a scheme that minimizes the total length of the original message and the appended signature. This line of research was motivated by several postal services interested by stamping machines capable of producing digital signatures. Although several message recovery schemes exist, their security is questionable. This paper proposes variants of DSA and ECDSA allowing partial recovery: the signature is appended to a truncated message and the discarded bytes are recovered by the verification algorithm. Still, the signature authenticates the whole message. Our scheme has some form of provable security, based on the random oracle model. Using further optimizations we can lower the scheme's overhead to 26 bytes with for a 2^{-80} security level, compared to forty bytes for DSA or ECDSA and 128 bytes 1024-bit RSA.

1 Introduction

Twenty years or so after the discovery of public key cryptography and digital signatures, the world appears ready for their large-scale deployment. Several signature schemes have been designed by the research community, either based on the celebrated RSA algorithm or on the discrete logarithm problem modulo a prime or over an elliptic curve. Standards have been crafted. Security proofs, notably using the so-called *random oracle model* have been proposed. Surprisingly, there still remain specific needs that appear in relation with some trading *scenarii* and which are not properly served by the current technology.

In some situations, it is desirable to use very short signatures; more accurately, one wishes to minimize the total length of the original message and the appended signature. In some respect, this is very similar to the problem one faces while trying to sign on a postcard without sacrificing too much of the (already limited) space available for the text. This analogy is not fortuitous: the motivation for short signatures has arisen from the needs of various postal services, which are currently investigating the possibility of integrating digital signatures into stamping machines. The space limitation here comes from the combined abilities of low-cost barcode printing machines and optical readers. Every byte one can save is of importance and the overhead of 128 bytes, implied by standard RSA signatures is not always acceptable. Even the forty byte overhead associated with DSA is hard to cope with using traditional (1-D) barcode technology.

1.1 1-D barcodes

Barcodes are alternating patterns of light and dark that encode specific information chunks. When scanned, barcodes can be converted back into the original string of text. Most barcodes consist of patterns of rectangles although some of the newer standards use other shapes. Barcodes can be scanned on the fly with little or no error under less than ideal conditions (*e.g.* folded envelops or damaged letters). The scanners that read barcodes emit a laser beam of a specific frequency that works by distinguishing the edges within a symbol allowing them to be scanned omnidirectionally. Each symbology (type of barcode) has unique start and stop bars (or some other unique pattern) that allows the scanner to discriminate between symbologies without human intervention. Most systems sacrifice one or more CRC digits to insure accuracy when scanned. Typical barcodes (such as Postnet, UPC, EAN, JAN, Bookland, ISSN or Code 39) have a capacity of a few bytes, normally up to thirty characters. A typical 1-D barcode is shown in figure 1.

Figure 1 : 1-D barcode.

Figure 2 : 2-D barcode.

Amongst the extensive bibliography about the 1-D barcodes available on-line, we particularly recommend [14]'s FAQ.

1.2 2-D barcodes

More sophisticated standards exist. These are based on two dimensional symbologies. Ordinary barcode is vertically redundant, meaning that the same information is repeated vertically. The heights of the bars can thus be truncated without any information loss. However, the vertical redundancy allows a symbol with printing defects, such as spots or voids to still be read. The higher the bars are, bigger is the probability that at least one path (horizontal section along the barcode) is still readable. A two dimensional (2-D) code stores information along the height as well as the length of the symbol (in fact, all human alphabets are 2-D codes). Since both dimensions contain information, at least some of the vertical redundancy is lost and error-correction techniques must be used to prevent misreads and produce acceptable read rates.

2-D code systems (for instance the PDF417 standard shown in figure 2) have become more feasible with the increased use of moving beam laser scanners and

CCD (charge coupled device) scanners. The 2-D symbol can be read with hand held moving beam scanners by sweeping the horizontal beam down the symbol.

Initially, 2-D symbologies were first applied to unit-dose packages in the healthcare industry. These packages were small and had little room to place a barcode. The electronics industry also showed an early interest in very high density barcodes, and 2-D symbologies since free space on electronics assemblies was scarce.

There are well over twenty different 2-D symbologies available today. Some look like multiple lines of barcodes stacked on top of each other and others resemble a honeycomb like-matrix. The reader can get a better idea of this diversity by consulting [13]. The capacity of 2-D codes is typically between a few hundreds to a couple of thousands of bytes.

1.3 Internet postage

More recently, the ability to encode a portable database has made 2-D symbologies attractive in postal applications: one example is storing name, address and demographic information on direct mail business reply cards. A good direct mail response is often less than two percent. If the return card is only coded with a *license plate*, the few replies must be checked against a very large database, perhaps millions of names. This can be quite expensive in computer time. If all the important information is printed in 2-D code at the time the mailing label is printed, there is very little additional cost, and a potential for great savings when the cards are returned. Similar savings can occur in field service applications where servicing data is stored in a 2-D symbol on equipment. The field engineer uses a portable reader to get the information rather than dialing up the home office's computer.

Figure 3 : Internet Postage.

In 1998, The United States Postal Service (USPS) introduced a new form of postage : Internet postage. Internet Postage is a combination of human-readable information and a 2-D barcode. To help the post office protect against fraud, the 2-D barcode contains information about the mail piece including the destination zip code, amount of postage applied, date and time the envelope was posted and a digital signature so that the post office can validate the authenticity of the postage.

Several companies were certified to distribute Internet postage (*e.g.* Pitney Bowes, Stamps.com *etc.*). In practice, such operators run postage servers that communicate with the USPS. When customers log on such a server, they can print Internet postage directly onto envelopes and labels using an ordinary laser or inkjet printer. A typical final result is shown in figure 3.

1.4 Short signatures

Although message recovery techniques seem to solve the signature size problem, they still suffer from several drawbacks. Firstly, they usually deal with messages of fixed length and it is unclear how to extend them when the message exceeds some given size. For example, the Nyberg-Rueppel scheme described in [6] applied to "redundant" messages of twenty bytes. This presumably means ten bytes for the message and ten for the redundancy but what if the message happens to be fourteen bytes long? Secondly, their security is not well understood. This is even an understatement: recently, a flaw has been found in the ISO/IEC 9796-1/2 standards (see [5, 4]). While completing this paper, we have been informed that Abe and Okamoto had independently investigated the matter and proposed a message recovery scheme proven secure in the random oracle model (see [1]). Still, they do not address the format question.

In this paper, we propose variants of DSA and ECDSA allowing partial recovery. The signature is appended to a truncated message and the discarded bytes are recovered by the verification algorithm. Still, the signature (which somewhat behaves as an error-correcting code) authenticates the whole message. Furthermore, we offer some form of proof for our scheme, based on the random oracle model. More accurately, the proof applies to a version of the scheme that slightly departs from the DSA/ECDSA design. Should closer compatibility with the standard be desired, one has to go over to a weaker security model (namely the so-called *generic* model). Still, this model gives strong evidence that the scheme's design is indeed sound.

Our scheme allows to recover ten bytes of the message with a security level 2^{-80}. This reduces the overhead of DSA/ECDSA signatures to thirty bytes. Further optimizations lower this figure to 26 bytes while keeping the same security level. They use several tricks such as transmitting additional bytes of the message as a subliminal part of the signature or slightly truncating the signature. This is traded-off against heavy (but still perfectly acceptable) preprocessing during signature generation and a slight increase of the verification time.

This paper focuses on signatures, not on certificates. We are perfectly aware that many trading *scenarii* will require appending a certificate to the signature and that the resulting overhead should be considered. For this reason, the size of the public key matters and the choice of elliptic curve signatures has been advocated in this context. Accordingly, we have chosen to describe our results in the elliptic curve setting. However, it is only the shorter length of the public key that makes EC signatures more attractive in terms of size. If the public key is known to the verifier, then, ordinary discrete logarithm signatures such as DSA are strictly equivalent (as far as size is concerned) to their EC analogs. In particular all our techniques go through, *mutatis mutandis*, when ordinary DL signatures are considered and the same optimizations in size that we suggest for EC signatures will equally apply to DL ones.

We close this introduction by briefly describing the organization of the paper: we first review the random oracle model and explain what kind of security it may provide; then, we introduce our partial recovery scheme and assess its soundness. Finally, we describe two possible optimizations and evaluate their cost in terms of memory requirement and computing time.

2 The random oracle model

2.1 The basic paradigm

The random oracle paradigm was introduced by Bellare and Rogaway in [2] as a practical tool to check the validity of cryptographic designs. It has been used successfully by Bellare and Rogaway ([3]) in connection with RSA signatures and by Pointcheval and Stern ([11]) to prove the security of El Gamal signatures. The model replaces hash functions by truly random objects and provides probabilistic security proofs for the resulting schemes, showing that attacks against these can be turned into efficient solutions of well-known mathematical problems such as factoring, the discrete logarithm problem (DL) or the ECDL problem.

Although the random oracle model is both efficient and useful, it has received a lot of criticism. It is absolutely true that proofs in the random oracle model are not proofs: they are simply a design validation methodology capable of spotting defective or erroneous designs when they fail. Besides, we will freely use the random oracle model in the context of DSA-like signatures. As is known, DSA uses for the generation of each signature a randomly chosen one-time key-pair $\{u, v\}$, with $v = g^u \bmod p$ (with standard notations) and derives a part of the signature c by considering v as an integer and reducing it modulo r. Similarly, ECDSA generates a random one-time key-pair $\{u, V\}$ (where V is a point on the elliptic curve defined by $V = u.G$), encodes V as an integer i and computes $c = i \bmod r$, where r is the order of G. As usual, the curve and the base point G are elements of the key. To provide proofs or spot design errors, we will replace the function $v \longrightarrow c$, and similarly the function $V \longrightarrow c$ by a random function R with range $[0, r-1[$. Practically, this can be achieved by hashing the encoding of v or V using a standard hash function such as SHA-1 [9]. Still, we do not necessarily suggest to hash the encoding. Of course this can be criticized in an even stronger way than the original paradigm underlying the random oracle model. For example, in DSA, we know that if v_1, v_2 are given, and if c_1, c_2 are their corresponding outputs, then $v_1 + v_2 \bmod p$ is exactly $(v_1 \bmod p) +$

$(v_2 \bmod p)$ or $(v_1 \bmod p) + (v_2 \bmod p) - p$ and therefore produces either the output $c_1 + c_2$ or $c_1 + c_2 - 1$ since r divides $p - 1$. Thus, the function $v \longrightarrow c$ is by no means random. Still, we note that it *seems* very difficult to control the value of v since it is produced by exponentiation and, accordingly, it is very difficult to distinguish c from an output drawn by a random function R. For this reason, we believe that random oracle proofs are still significant. In the next paragraph we give further arguments in support of the random oracle model by relating our approach to the so-called *generic algorithms* used by Shoup ([12]).

2.2 A note on generic algorithms

A *generic algorithm* is an algorithm that uses a group structure but can only handle the group elements by either calling arguments passed to the algorithm or by applying the group operations to previously accessed elements. The concept has been introduced by Nechaev ([8]) and has been successfully applied by Shoup ([12]) to the discrete logarithm problem and the Diffie-Hellman problem. Basically, it rules out techniques that would take advantage of the actual representation of the group elements. Typically, methods such as the Index-calculus, which try to factor elements of the group into small prime factors do not fall under the scope of generic algorithms. Similarly, any method that would process in any way the coordinates of an elliptic curve point would be beyond reach of generic algorithms. The interesting point is that no such method is known.

The concept of a generic algorithm is not easy to explain and we give our own definition, which is inspired by [12] while not being exactly similar. Any group element V receives a name $\hat{V}$. The mapping that assigns a name to an element is random and the algorithm can only access group elements by invoking their names. To compute $V + V'$ (or $V - V'$), the algorithm submits $\hat{V}$ and $\hat{V}'$ to a random oracle that returns a name for $V + V'$ (or $V - V'$). In such a model, the only way to compute an analog of the various functions $R(V)$ introduced in the previous section, is to use the random name $\hat{V}$. In other words, by considering that $R(V)$ is a random function, we are simply working in the generic model using $R(V)$ in place of $\hat{V}$. In essence, the mechanism is similar to the manipulation of data (V, V') using pointers $(\hat{V}, \hat{V}')$ and functions $(+, -)$.

3 The partial recovery scheme

3.1 Nyberg-Rueppel signatures

We say that a signature scheme allows *message recovery* if the message m is a deterministic function of the signature. Such signatures make it possible to avoid sending the message together with the signature. However, one should be very careful since such schemes are inherently subject to forgeries. In other words, some redundancy should be added to the message.

A DSA-like signature with message recovery has been considered by Nyberg and Rueppel ([10], hereafter NR) and an ECDSA variant of this scheme, included in [6], is described in figure 4.

Signature	
	1. generate a random key-pair $\{u, V\}$
	2. form f from m by adding the proper redundancy
	3. encode V as an integer i
	4. $c \leftarrow i + f \bmod r$
	5. if $c = 0$ go to step 1
	6. $d \leftarrow u - sc \bmod r$
	7. output the pair $\{c, d\}$ as the signature
Verification	
	1. input a signature $\{c, d\}$
	2. if $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop
	3. $P \leftarrow d.G + c.W$
	4. if $P = \mathcal{O}$, output invalid and stop
	5. encode P as an integer i
	6. $f \leftarrow c - i \bmod r$
	7. if the redundancy of f is incorrect output invalid and stop
	8. output valid and the underlying message m

Figure 4 : Nyberg-Rueppel signatures (outline).

In the above, f is a *message with appendix*. It simply means that it has an adequate redundancy. The encoding mentioned in step 2 of figure 1 is defined in the standard. Its particular format is not important to us. Applying a hash function to this encoding consists of replacing step 2 by: "2. encode-and-hash V as an integer i".

Modified that way, the scheme can be proven secure in the random oracle model, with arguments very close to those used in the sequel. We will not undertake this task as we feel that NR signatures are not flexible enough for our purposes. Assuming that f consists of ten message bytes and ten redundancy bytes, NR is perfectly suitable for messages shorter than ten bytes but leaves unanswered the question of dealing with messages of, say, fifteen bytes.

3.2 An ECDSA variant with partial recovery

There are numerous ways to modify the NR design in order to achieve *partial message recovery*. In this section, we propose a possible choice that is as close as possible to the original ECDSA. A similar construction, that we omit, applies to the regular DSA.

Our proposal allows to sign a message $m = m_1 || m_2$, where $||$ denotes concatenation and to only transmit m_2 together with the signature. The partial message recovery concept is, of course, not new; the RSA-oriented iso 9796-2 standard [7] specifies explicitly two recovery modes (total and partial) but to the best of our knowledge, this notion was never extended to the DLP context. We propose to sign m, using the algorithm described in figure 5 where H denotes any standard hash function such as SHA-1.

Signature

1. generate a random key pair $\{u, V\}$
2. form f_1 from m_1 by adding the proper redundancy
3. encode-and-hash V as an integer i
4. $c \leftarrow i + f_1 \bmod r$
5. if $c = 0$ go to step 1
6. $f_2 \leftarrow H(m_2)$, $d \leftarrow u^{-1}(f_2 + sc) \bmod r$
7. if $d = 0$ go to step 1
8. output the pair $\{c, d\}$ as the signature

Verification

1. input a signature $\{c, d\}$ and a partial message m_2
2. if $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop
3. $f_2 \leftarrow H(m_2)$, $h \leftarrow d^{-1} \bmod r$, $h_1 \leftarrow f_2 h \bmod r$
4. $h_2 \leftarrow ch \bmod r$, $P \leftarrow h_1.G + h_2.W$
5. if $P = \mathcal{O}$ output invalid and stop
6. encode-and-hash P as an integer i
7. $f_1 \leftarrow c - i \bmod r$
8. if the redundancy of f_1 is incorrect output invalid and stop
9. output valid and the underlying message m_1

Figure 5 : Partial recovery signatures (outline).

Note that we do not necessarily advocate our encode-and-hash paradigm. Replacing *encode-and-hash* by *encode* in the above yields a scheme that is more closely modeled after ECDSA. Still, even if it remains significant, the security proof has a weaker status as explained in section 2.

3.3 Security proof

We use the random oracle model to provide evidence in favor of the security of the new scheme. We will thus assume that the function $R(V)$ which encodes the point V as an integer i and computes $i \bmod r$ is random. Finally, we will assume that the probability ϵ that a random element f of $[0, r-1]$ has the expected redundancy is very small. Basically, we want to show that an adversary who can forge a message/signature pair with probability $\epsilon + \alpha$ significantly above ϵ can be used to solve the ECDL problem with non-negligible probability. This is along the lines of [11]. However, we will not be careful about the security estimates for we only wish to support the correctness of our design.

Referring to the scheme described in figure 5, we let $\mathcal{A}$ be an attacker able to forge a pair consisting of a message $m = m_1 || m_2$ and a signature $\{c, d\}$ with a success probability $\geq \epsilon + \alpha$. We consider the queries asked to the oracles as ordered lists and let j and k be the respective indices corresponding to the time when P and m_2 are respectively queried from the R-oracle and the H-oracle, during the computation of $\mathcal{A}$. If j or k does not exist, we set $j = \infty$ or $k = \infty$. Similarly, we let δ be the truth-value of the statement "P is queried before m_2", where the truth value is one if neither question is asked.

By standard arguments from [11], we see that there is a set of triples A such that:

i) A has probability $\geq \alpha/2$

ii) For any $\{j, k, \delta\}$ the conditional probability of success of $\mathcal{A}$ when P is queried at j, H queried at k and the statement "P is queried before m_2" has value δ is $\geq \epsilon + \alpha/2$.

We first claim that no triple $\{j, k, \delta\}$ in A can have an infinite value. Assume that $j = \infty$. Checking the signature precisely corresponds to computing $i = R(P) \bmod r$ and verifying that $c - i \bmod r$ has the proper redundancy. Now, if R is controlled by a random oracle, and if P has not been queried during the computation performed by $\mathcal{A}$, then, $R(P)$ can be any value and the test will fail with probability $1 - \epsilon$. From this, we may infer that the conditional success probability corresponding to the triple cannot be $\geq \epsilon + \alpha/2$. We turn to the case $k = \infty$. If the value of H at m_2 has not been queried by $\mathcal{A}$ during its computation, then, it is only computed at the verification step and, again, with probability $\geq 1 - \epsilon$, the resulting value of P differs from values queried to the R-oracle.

We now apply the forking lemma from [11] by playing the attacker a first time and generating a replay attack as explained below. Note that, with probability $\geq \alpha/2$, the triple $\{j, k, \delta\}$ corresponding to the first execution belongs to A, in which case neither j nor k is infinite.

We now distinguish two cases depending on the value of δ :

• If $\delta = 0$, then m_2 is queried before P. We apply the forking technique at P and obtain, by a replay attack, another signature pair $m' = m'_1 || m'_2$, $\{c', d'\}$. From the facts that both computations are similar until P is queried we infer that $m'_2 = m_2$ and that

$$P = h_1.G + h_2.W = h'_1.G + h'_2.W$$

Equivalently

$$(f_2 d^{-1}).G + (cd^{-1}).W = (f'_2 d'^{-1}).G + (c'd'^{-1}).W$$

From the first equality, we obtain $f_2 = f'_2$ and from the second

$$f_2(d' - d).G = (c'd - cd').W$$

This discloses the secret logarithm of W in base G unless $cd' - c'd$ vanishes, in which case $f_2(d - d')$ also vanishes. Observe that f_2 which has been queried from H is non zero with overwhelming probability. Thus, the secret key has been found, except if $d = d'$. Since d is non zero, this implies $c = c'$, which reads $i + f_1 = i' + f_1'$, where i and i' are the respective answers of the R-oracle to the P question. Due to the redundancy of f', this can only happen with probability $\leq \epsilon$. Since the conditional probability of success at $\{j, k, \delta\}$ is $\geq \epsilon + \alpha/2$, the replay discloses the discrete logarithm of the public key with probability $\geq \alpha/2$ (once we know that $\{j, k, \delta\}$ lies in A).

- If $\delta = 1$ we fork at the point where m_2 is queried. We obtain a second message-signature pair $m' = m_1' || m_2'$, $\{c', d'\}$ and, this time, we note that $i = i'$, since the answer of the R-oracle to the P query is similar and, again, that

$$P = h_1.G + h_2.W = h_1'.G + h_2'.W$$

We get

$$(f_2 d' - f_2' d).G = (c'd - cd').W$$

From this, we can compute the discrete logarithm of W in base G unless $c'd - cd'$ and $f_2 d' - f_2' d$ both vanish modulo r. To complete the security proof as above, we only have to see that this exceptional case can only happen with probability $\leq \epsilon$. Indeed, if it actually happens, we have

$$c'd = cd' \bmod r$$

$$f_2 d' = f_2' d \bmod r$$

from which we get

$$f_2 cd' = f_2 c'd = f_2' cd \bmod r$$

and, since d is not zero

$$f_2 c' = f_2' c \bmod r$$

which gives

$$f_2(f_1' + i) = f_2'(f_1 + i) \bmod r$$

and, finally, taking into account the fact that f_2, queried from R is non zero with overwhelming probability

$$f_1' = f_2' f_2^{-1}(f_1 + i) - i \bmod r$$

Since f_2' is randomly chosen by the H-oracle, f_1' has the requested redundancy with probability $\leq \epsilon$. This completes the proof.

3.4 Adaptive attacks

In the previous proof, we have considered the case of an attacker forging a message-signature pair from scratch. In more elaborate *scenarii* an attacker may adaptively request signatures corresponding to messages of his choice. In other words, the attacker, modeled as a machine, interacts with the legitimate signer by submitting messages that are computed according to its program.

We show how to modify the security proof that was just given to cover the adaptive case. We have to explain how to turn the attacker into a machine that discloses the logarithm of a given element W in base G. Basically, we wish to use the attacker in the same way and apply the forking technique. The main difficulty comes from the fact that we have to mimic the signer's action without knowing the secret key.

To simulate the signer when he has to output the signature of a message $m = m_1||m_2$, we pick the signature $\{c, d\}$ at random, query the H-oracle at m_2 and compute the point

$$V = (f_2 d^{-1}).G + (cd^{-1}).W$$

with $f_2 = H(m_2)$. Next, we "force" the R-oracle to adopt c at its value at V. Since c has been chosen randomly, this does not produce any noticeable difference unless the same V is forced to two different values. It can be checked that this happens with negligible probability.

3.5 Practical consequences

Thus, we have shown, in the random oracle model, that an attacker can be turned into an algorithm that solves the ECDL problem. This establishes the soundness of the new design, provided that the probability ϵ attached to the redundancy is small enough. From a practical standpoint, the only attack suggested by the above analysis consists in picking the signature $\{c, d\}$ at random, generating a message m_2, computing the hash value $f_2 = H(m_2)$ and applying the message recovery algorithm, hoping that the resulting value of f_1, computed at step 7 has the correct redundancy. This strategy succeeds with a probability $\leq \epsilon$. Note that we have not used any assumption on the format of the redundancy, which can simply consist of a requested number of fixed leading or trailing bytes. Since the security level required for signatures is about 2^{80}, we recommend to take $\epsilon \leq 2^{-80}$. When signing messages with ℓ bytes, $\ell \geq 10$, the new design allows to only append to the signature $\{c, d\}$ a part of the message m_2 which is $\ell - 10$ bytes long. The rest of the message m_1 is recovered by the verification algorithm.

4 Bandwidth optimizations

We now investigate possible optimizations of our scheme that allow to save a few extra bytes. We use two different tricks:

1. transmitting additional message bytes as a subliminal part of the signature, by suitably choosing the random part during signature generation.
2. truncating the signature, leaving completion to be performed during the verification phase.

Of course, both suggestions increase the time complexity of signature generation (in the first case) or verification (in the second case). For this reason, we cannot expect to gain too many bytes per trick. Still, we show that it is quite reasonable to squeeze three bytes out of the first trick by using some form of preprocessing and one extra byte from the second.

There are many ways in which the above ideas can be applied; bytes of the message can be embedded into c, d or i. Similarly, either c or d can be truncated. We will only cover the case where i is used for subliminal information and d is truncated. The rest is left to the reader.

4.1 Embedding bytes into i

Assume that one wishes to embed ℓ bytes of m in i, where ℓ is a small integer. For example, assume that we try to stuff these bytes into the trailing part of i. One would then repeat the first steps of the signature generation algorithm until a correct value of i appears, *i.e.* an i whose trailing bytes match the given ℓ bytes of the message. Clearly, this is possible only if ℓ is small and yields the scheme presented in figure 6 that allows to sign a message $m = m_1||m_2$, where m_1 has $10 + \ell$ bytes and to only transmit m_2. The security proof of section 3.3 goes through, word for word, for the modified scheme.

4.2 Preprocessing

Preprocessing appears very helpful in relation with the optimization described in the previous section. Basically, one should store pairs $\{u, i\}$ and access these pairs by the value of $i \bmod 2^{8\ell}$. Signature generation might fail if the table's list of elements is empty at some ℓ byte location. Thus, it is important to keep a sufficiently large number τ of elements for each ℓ byte values and to refresh the table regularly.

The size of the table is $\simeq 40\tau 2^{8\ell}$ bytes; $\ell = 3$ corresponds to 640τ Mbytes which is quite acceptable; $\ell = 4$ goes up to 160τ Gbytes, which appears too much. Note that ℓ is not necessarily an integer: bytes can be cut into nibbles and $\ell = 3.5$ could also be considered (10τ Gbytes).

4.3 Truncating d

We now turn to the second optimization suggested above. It consists in truncating k signature bytes. For example, one could omit the k trailing (or leading) bytes of c. This basically means issuing 2^{8k} candidate signatures. The correct signature is spotted at signature verification: only the correct choice is accepted by the verification algorithm.

It is easily seen that the security of the truncated signature is closely related to the security of the original scheme. An attacker able to forge a truncated signature will complete his forgery to an actual signature by using the verification algorithm. Thus, the only difference is the verifier's workload.

Signature

1. generate a random key pair $\{u, V\}$
2. discard the ℓ trailing bits of m_1
3. form f_1 from the result m_1' by adding the proper redundancy
4. encode-and-hash V as an integer i
5. $c \leftarrow i + f_1 \bmod r$
6. if $c = 0$ or $i \neq m_1 \bmod 2^{8\ell}$ go to step 1
7. $f_2 \leftarrow H(m_2)$, $d \leftarrow u^{-1}(f_2 + sc) \bmod r$
8. if $d = 0$ go to step 1
9. output the pair $\{c, d\}$ as the signature

Verification

1. input a signature $\{c, d\}$ and a partial message m_2
2. if $c \notin [1, r-1]$ or $d \notin [1, r-1]$, output invalid and stop
3. $f_2 \leftarrow H(m_2)$, $h \leftarrow d^{-1} \bmod r$, $h_1 \leftarrow f_2 h \bmod r$
4. $h_2 \leftarrow ch \bmod r$, $P \leftarrow h_1.G + h_2.W$
5. if $P = \mathcal{O}$, output invalid and stop
6. encode-and-hash P as an integer i
7. $f_1 \leftarrow c - i \bmod r$
8. if the redundancy of f_1 is incorrect output invalid and stop
9. append to m_1' the ℓ trailing bytes of i
10. output valid and the underlying message m_1

Figure 6 : Partial recovery signatures (outline).

At first glance, it seems that, in order to check truncated signatures, the verifier will have to verify 2^{8k} signatures, which appears prohibitive even for $k = 1$. However, optimizations are possible since the various elliptic curve points that the verifier should compute are

$$P = h_1.G + h_2.W$$

where only $h_2 = cd^{-1} \bmod r$ depends on c. Let c_0 be the completion of the truncated value of c by zeros. Writing P as

$$P_j = h_1.G + c_0 d^{-1}.W + jd^{-1}.W$$

we see that the verification algorithm can be organized as follows:

1. $Z \leftarrow d^{-1}.W$
2. $P \leftarrow P_0 + c_0.Z$
3. while a correct signature has not been found $P \leftarrow P + Z$

Considering that c, d are 160 bit integers and that a standard double-and-add algorithm is used, one can estimate the number of elliptic curve operations needed to compute P_0 as close to 240. Z and P_0 can be simultaneously computed in about 320 additions by sharing the "double" part. Finally, step 3 is expected to require 128 extra additions. For $k = 1$, the overhead does not exceed the verification time needed of a regular signature.

There is a trick which slightly improves performances: instead of producing the signature as $\{c, d\}$, one can produce $\{h_2, d\}$, with $h_2 = cd^{-1} \bmod h$. Truncating h_2 yields slightly better computational estimates.

5 Conclusion

We have shown how to minimize the overall length of an elliptic curve signature *i.e.* the sum of the lengths of the signature itself and of the message (or part of the message) that has to be sent together with the signature. Up to thirteen message bytes can be recovered in a secure way from a signature and an additional one-byte saving on the signature itself can be achieved.

The proposed schemes have been validated by a proof in the random oracle model and can therefore be considered sound. All our schemes have ordinary discrete logarithm analogs.

6 Acknowledgments

The authors are grateful to Jean-Sébastien Coron for his help and comments. We also thank Holly Fisher for sending us figure 3. Stamps.com's Internet Postage system (`http://www.stamps.com`) is covered by Stamps.com Inc. copyright (1999). We underline that the image is only given for illustrative purposes and that this specific system does not implement the signature scheme proposed in this paper.

References

1. M. Abe and T. Okamoto, *A signature scheme with message recovery as secure as discrete logarrithms*, Proceedings of Asiacrypt'99, LNCS, Springer-Verlag, to appear, 1999.
2. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, Proceedings of the 1-st ACM conference on communications and computer security, pp. 62–73, 1993.
3. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*, Proceedings of Eurocrypt'96, LNCS 950, Springer-Verlag, pp. 399–416, 1996.
4. D. Coppersmith, S. Halevi and C. Jutla, ISO *9796-1 and the new forgery strategy.*, manuscript, July 28, 1999.
5. J.-S. Coron, D. Naccache and J.P. Stern, *On the security of RSA padding*, Proceedings of Crypto'99, LNCS 1666, Springer-Verlag, pp. 1–18, 1999.
6. IEEE P1363 Draft, *Standard specifications for public key cryptography*, (available from `http://grouper.ieee.org/groups/1363/index.html`), 1998.
7. ISO/IEC 9796-2, *Information technology - Security techniques - Digital signature scheme giving message recovery, Part 2 : Mechanisms using a hash-function*, 1997.
8. V.I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm.* Mathematical Notes, 55(2), pp. 165–172, 1994. Translated from *Matematicheskie Zametki* 55(2), pp. 91–101, 1994.
9. National Institute of Standards and Technology, *Secure hash standard*, FIPS publication 180-1, April 1994.
10. K. Nyberg and R. Rueppel, *A new signature scheme based on the DSA, giving message recovery*, Proceedings of the 1-st ACM conference on communications and computer security, pp. 58–61, 1993.
11. D. Pointcheval and J. Stern, *Security proofs for signature schemes.* Proceedings of Eurocrypt'96, LNCS 950, Springer-Verlag, pp. 387–398, 1996.
12. V. Shoup, *Lower bounds for discrete logarithms and related problems.* Proceedings of Eurocrypt'97, LNCS 1233, Springer-Verlag, pp. 256–266, 1997.
13. `http://www.adams1.com/pub/russadam/stack.html`
14. `http://www.azalea.com`

This article was processed using the LaTeX macro package with LLNCS style