Digital Signatures Do Not Guarantee Exclusive Ownership

Thomas Pornin and Julien P. Stern

Cryptolog International, Paris, France
thomas.pornin@cryptolog.com and julien.stern@cryptolog.com

Abstract. Digital signature systems provide a way to transfer trust from the public key to the signed data; this is used extensively within PKIs. However, some applications need a transfer of trust in the other direction, from the signed data to the public key. Such a transfer is cryptographically robust only if the signature scheme has a property which we name *exclusive ownership*. In this article, we show that the usual signature algorithms (such as RSA[3] and DSS[4]) do *not* have that property. Moreover, we describe several constructs which may be used to transform a signature scheme into another signature scheme which provides exclusive ownership.

1 Introduction

Digital signature schemes based on public-key cryptography are now used in many communication protocols. Signatures are used to convey trust *from* a public key *to* the data which is signed: if the public key is known (by some other mean) to be associated with some entity who "owns" it (i.e., the entity has exclusive access to the corresponding private key), then a valid signature on some data "proves", in a way verifiable by third parties and non repudiable by the key owner, that the key owner had access to the data and deliberately agreed to that association between his public key and the data. This assumes, of course, that no other entity than the key owner has access to the private key, and that the signature and verification algorithms are uncrackable with today's technology. Various semantics can be attached to the signature; PKIs use it as a way to *certify* that the data is correct (the key owner formally guarantees the exactness of the data).

In this article, we are interested in the "reverse" problem, in which a signature on some data is known, and we want to know whether the existence of a public key which validates that signature implies that the corresponding private key was used to produce the signature in the first place. It so happens that usual signature algorithms such as RSA[3] and DSS[4] do *not* provide such guarantee; however, some applications, such as the certificate revocation process through CRLs in X.509[1], need that property. This problem has already been partially studied in [5].

We will first define, informally and then formally, what is the actual property that we are interested in; we call it *exclusive ownership* and we will define several subdivisions. Then we will describe in detail how some widely used algorithms do not provide exclusive ownership. A further section will show how this can be fixed.

This work was partially supported by the French Ministry of Research RNRT X-CRYPT project.

1.1 Informal Definitions

We informally consider a signature scheme. An entity A owns a key pair which we note $(K^A_{\mathrm{pub}}, K^A_{\mathrm{priv}})$ (the public key and the private key, respectively). We suppose that the signature scheme has the usual properties which make that we deem the scheme secure (i.e., it is computationaly infeasible, without knowledge of the private key, to compute existential forgeries[6]).

We now consider an external entity B which has no knowledge of the private key K^A_{priv}. B has access to a set of triplets $(K^A_{\mathrm{pub}}, m, s)$ where K^A_{pub} is A's public key, m is some binary message, and s is a valid signature of m relatively to K^A_{pub}. B will then try to produce a new triplet, which is accepted by the verification algorithm, and where some or all of the three triplet components are not part of the set of components issued by A. In other words, A has signed, using its (unique) private key, a set of distinct messages, and produced the corresponding signatures. B wants to produce a triplet comprising a public key, a message and a signature, such that the public key is distinct from A's public key, or the message is not one of those signed by A, or the signature is not one of those computed by A, or any combination of some of those properties. Moreover, we require the following properties:

– when the public key produced by B is not equal to A's public key, B knows the corresponding private key;
– when both the message m and the signature s are kept from the values issued by A, s must be a valid signature for m relatively to K^A_{pub} (in other words, if B keeps one of A's messages and one of A's signatures, the signature must apply to that message).

The figure 1 explicits the eight possible types of triplets that B may attempt to produce. In that figure, a triplet issued by B may contain:

– either A's public key K^A_{pub}, or a different public key K^B_{pub} for which B knows the corresponding private key K^B_{priv};
– either one of the messages m signed by A, or a new message m';
– either one of the signatures s issued by A, or a new signature s'.

Transforms 1, 6 and 8 are trivial:

– for transform 1, B just selects one of the triplets issued by A;
– for transform 6, B creates its own key pair using the normal procedure, and then signs one of the messages issued by A;
– for transform 8, B creates its own key pair and signs an entirely different message.

Fig. 1. Signature transforms

$$\begin{tikzcd}
\boldsymbol{K^{B}_{\mathrm{pub}}},\ \boldsymbol{m'},\ \boldsymbol{s'} & K^{A}_{\mathrm{pub}},\ m,\ s & K^{A}_{\mathrm{pub}},\ m,\ \boldsymbol{s'} \\
\boldsymbol{K^{B}_{\mathrm{pub}}},\ \boldsymbol{m'},\ s & (K^{A}_{\mathrm{pub}}, m, s)* \arrow[l, "7"'] \arrow[r, "3"] \arrow[u, "1"'] \arrow[ur, "2"] \arrow[ul, "8"] \arrow[d, "5"] \arrow[dl, "6"] \arrow[dr, "4"] & K^{A}_{\mathrm{pub}},\ \boldsymbol{m'},\ s \\
\boldsymbol{K^{B}_{\mathrm{pub}}},\ m,\ \boldsymbol{s'} & \boldsymbol{K^{B}_{\mathrm{pub}}},\ m,\ s & K^{A}_{\mathrm{pub}},\ \boldsymbol{m'},\ \boldsymbol{s'}
\end{tikzcd}$$

Transforms 3 and 4 are computationaly infeasible for any good signature scheme: B must not be able to produce a new message and a signature which will be accepted by the verification algorithm relatively to A's public key. When either 3 or 4 is possible, it is called a *forgery attack*[6].

Transform 2 is subject to debate. For any given message m, that several distinct signatures are accepted relatively to the same public key is normal if the signature algorithm is randomized, which is a desirable property. If B can apply transform 2 without knowing the private key K^{A}_{priv}, the signature scheme is said to be *malleable*[7]. Whether malleability is acceptable for a signature scheme depends on the context. This paper does not deal any further with that question.

Transforms 5 and 7 are the purpose of this article. We define a new property which we call *exclusive ownership.* A signature scheme will be said to have conservative or destructive exclusive ownership if it is computationaly infeasible for B to apply, respectively, transform 5 or 7.

We will now define exclusive ownership formally.

1.2 Formal Definitions

Let us consider a signature scheme; it consists of the following three algorithms (as described in [6]):

- a *key generation algorithm* G which, on input 1^k (where k is the "security parameter"), outputs a pair $(K_{\mathrm{pub}}, K_{\mathrm{priv}})$ (a public key and a corresponding private key);
- a *signature algorithm* Σ which, using the private key K_{priv}, produces for a message m (in the space of messages $\mathcal{M}$) a signature $s = \Sigma(K_{\mathrm{priv}}, m)$ (in the space of signatures $\mathcal{S}$);
- a *verification algorithm* V which takes as input a public key K_{pub}, a message m and a signature s, and returns **true** if the signature is "valid", **false** otherwise.

When K_{pub} and K_{priv} are such that, for every possible message m in $\mathcal{M}$, we have $V(K_{\text{pub}}, m, \Sigma(K_{\text{priv}}, m)) = \textbf{true}$, then we say that K_{priv} *matches* K_{pub}. G must be such that it outputs only matching key pairs.

We now suppose that there exists an entity which has run the algorithm G and thus owns a key pair $(K_{\text{pub}}, K_{\text{priv}})$. That entity then proceeds with the signing of t messages m_1, m_2,... m_t using K_{priv}, thus producing s_1, s_2,... s_t. Another entity, the "attacker", will try to produce a new key pair $(K'_{\text{pub}}, K'_{\text{priv}})$, a message m' and a signature s', such that:

– K'_{priv} matches K'_{pub};
– $V(K'_{\text{pub}}, m', s') = \textbf{true}$;
– s' is one of the $s_i (1 \leq i \leq t)$; we note it s_j.

Definition 1. *The signature scheme is said to provide* conservative exclusive ownership *(CEO) if it is computationaly infeasible for the attacker, given* K_{pub} *and* t *pairs* (m_i, s_i) *such that* $V(K_{\text{pub}}, m_i, s_i) = \textbf{true}$ *for* $1 \leq i \leq t$*, to produce values* K'_{pub}*,* K'_{priv} *and* m' *such that* $K'_{\text{pub}} \neq K_{\text{pub}}$*,* K'_{priv} *matches* K'_{pub}*,* $m' = m_j$ *for some value* j *(*$1 \leq j \leq t$*) and* $V(K'_{\text{pub}}, m_j, s_j) = \textbf{true}$*.*

Definition 2. *The signature scheme is said to provide* destructive exclusive ownership *(DEO) if it is computationaly infeasible for the attacker, given* K_{pub} *and* t *pairs* (m_i, s_i) *such that* $V(K_{\text{pub}}, m_i, s_i) = \textbf{true}$ *for* $1 \leq i \leq t$*, to produce values* K'_{pub}*,* K'_{priv}*,* m' *and* s' *such that* $K'_{\text{pub}} \neq K_{\text{pub}}$*,* K'_{priv} *matches* K'_{pub}*,* $s' = s_j$ *for some value* j *(*$1 \leq j \leq t$*),* $m' \neq m_j$ *and* $V(K'_{\text{pub}}, m', s_j) = \textbf{true}$*.*

Definition 3. *The signature scheme is said to provide* universal exclusive ownership *(UEO) if it is computationaly infeasible for the attacker, given* K_{pub} *and* t *pairs* (m_i, s_i) *such that* $V(K_{\text{pub}}, m_i, s_i) = \textbf{true}$ *for* $1 \leq i \leq t$*, to produce values* K'_{pub}*,* K'_{priv}*,* m' *and* s' *such that* $K'_{\text{pub}} \neq K_{\text{pub}}$*,* K'_{priv} *matches* K'_{pub}*,* $s' = s_j$ *for some value* j *(*$1 \leq j \leq t$*) and* $V(K'_{\text{pub}}, m', s_j) = \textbf{true}$*.*

Universal exclusive ownership naturally implies conservative and destructive exclusive ownership. However, there are no other implications. See section 3 for further details.

We call *second key construction* the process of defeating exclusive ownership, that is building a quartet $(K'_{\text{pub}}, K'_{\text{priv}}, m', s_j)$ which verifies the properties described above. Some distinctions can be made, depending on whether and when B chooses the messages that A signs (known messages, chosen messages and adaptative chosen messages second key construction).

1.3 Relevance to X.509 Revocation

X.509[1] is a standard for public-key infrastructures. Public keys owned by entities are *certified* by encapsulating them into certificates, which are digitally signed by certification authorities, whose public keys are themselves published within other certificates. A certificate establishes a binding between an identity and a public key. Certificate validation is the process of verifying those signatures, from an a priori trusted public key (often called a "root authority" or "trust anchor") downto an end-entity certificate. Validation uses signatures in the "proper" way: trust is transfered from knowledge of the public key to the signed data.

X.509 certificates can be revoked for a number of reasons, including private key compromise and normal rights management. Revocation is a way of externally "cancelling" a certificate, by declaring its signature invalid. The most common way to revoke a certificate is to publish a Certificate Revocation List (CRL) which contains the revocation status of the certificates issued by an authority. The paradigm here is that a given certificate has a unique issuer, and a unique revocation status. Applications use this unicity to implement caching strategies which prevent certificate validation from using too many computing resources.

The problem is that the validator considers a CRL as valid for a given certificate if it is signed relatively to the same public key than the certificate itself. If the signature algorithm lacks exclusive ownership, then a rogue authority could create a fake certificate issuer, with a constructed second key which validates the certificate. The rogue authority could then publish another CRL. From the point of view of the validator, the certificate has two issuers, and two relevant CRLs (both fulfill all validation procedures, including signature verification). The validation algorithm is then unsound, because it will return a result which depends upon which issuer is tried first by the validator. Moreover, for an application which uses a cache for certificate revocation status, the rogue authority can "contaminate" other certificate hierarchies, by declaring them as revoked, even though these certificates are not part of the authority subtree. This is in contradiction with the principle of a hierarchical PKI.

A detailed analysis of this weakness is presented in [2].

2 Usual Algorithms Lack Exclusive Ownership

The most widely used signature algorithms do not have the exclusive ownership property, neither conservative nor destructive. For some of them, we present here second key construction algorithms which work in the most adverse situations (only one signature is available on a known message, and the attacker arbitrarily chooses the message that will be validated by the signature relatively to the new public key).

2.1 RSA

Algorithm Description

RSA is a signature algorithm[1] whose principle was first described in [8]. The core operation is a modular exponentiation:

– the public key is (n, e) where n is a (big) composite number and e is an arbitrary value, usually small, which is relatively prime to $\phi(n)$;

[1] RSA is also an asymmetric encryption algorithm, but we consider here only the signature version.

- the private key is a number d;
- a value v (modulo n) is signed by computing $s = v^d \bmod n$;
- a signature s is verified by computing $w = s^e \bmod n$ and checking that $w = v$.

The private key d must be such that the verification algorithm works; it can be selected such that $ed = 1 \bmod \phi(n)$ (other private key values are possible, but this distinction is not important in this paper).

The core algorithm signs a value modulo n; several standards have been defined, which describe how an arbitrary binary message m is transformed into a value v in a way which makes RSA a secure signature scheme (this is called a *padding scheme* because it was historically done by concatenating the hashed message with some specific data). The most widely used standard is PKCS#1[3], which contains both a deterministic (old version, so-called "1.5") and a randomized (new version, named PSS) padding schemes. The resistance of these schemes to forgeries remains unchallenged; PSS even features strong security proofs.

Second Key Construction

Whatever the padding scheme used, as long as it does not use the value of n but only its size (which is the case for PKCS#1 padding schemes), RSA does not have the exclusive ownership property. We consider a public key (n, e) and a signature s on a message m; s is a positive integer strictly lower than n. We now consider another message m' which the padding scheme transforms into a value v' modulo n. We build a new public key (n', e') with the following process:

1. We select distinct random numbers q_i such that:
   - q_i is prime;
   - $p_i = 2q_i + 1$ is prime;
   - p_i is relatively prime to v' and s;
   - there exists some value d_i such that $v'^{d_i} = s \bmod p_i$ and d_i is relatively prime to $2q_i$ (this property is easily checked if q_i is small enough, e.g. by exhaustive search on d_i).
2. From the set of numbers p_i, we choose k of them (which we number from 1 to k), such that $n' = \Pi_{i=1}^{k} p_i$ has the appropriate size (n' must be greater than s but also small enough for v' to be a valid padding result with m' as message and n' as modulus).
3. We define the values e'_i to be such that $e'_i d'_i = 1 \bmod (p_i - 1)$.
4. By applying the Chinese remainder theorem[9], we compute values d' and e' which are such that $d' = d_i \bmod (p_i - 1)$ and $e' = e_i \bmod (p_i - 1)$ for all i from 1 to k.

This process yields n', d' and e' which are such that:

- (n', e') is a valid RSA public key and d' is a valid RSA private key matching that public key;
- $v'^{e'} \bmod n' = s$, which means that s is a valid signature of m' relatively to the public key (n', e').

This process is fast if the p_i are chosen such that the computation of each d_i value is fast. This is a discrete logarithm, which is easy if the p_i are small enough. The p_i values are chosen as $p_i = 2q_i + 1$ where q_i is prime, so that it is probable, for each such value p_i, that the value of d_i is relatively prime to $\phi(p_i) = p_i - 1 = 2q_i$. Besides, the primality of each q_i simplifies greatly the final reconstruction of d' and e' with the CRT.

A very naive implementation, in the Java programming language, on a simple and cheap PC, performs this second key construction in a few seconds for a 1024-bit original modulus n.

Remarks

• The message m' in the algorithm described above is chosen arbitrarily and may be equal to the message m. Thus, this algorithm demonstrates that RSA has neither CEO nor DEO.

• The algorithm described above produces an RSA modulus which is quite smooth: it is a product of small factors. The factors can be made bigger at the expense of a longer computation, to retrieve the values d_i. Detecting that the modulus is smooth depends on the size of the smallest of the p_i and implies a computational effort which is roughly of the same order than the one deployed by the attacker to compute each d_i. The attacker is therefore in a winning situation, since he usually has much more computing power than what any verifier is willing to spend before accepting a signature as valid.

• The produced public exponent e' is not controlled by the attacker, and as such is usually big. Most existing RSA public exponents are small (in order to speed up signature verification) and indeed some widely deployed implementations are not even able to handle public exponents bigger than 32 bits. Whether second key construction *with a small public exponent* is computationaly feasible is not currently known.

• Our intuition is that, with a fixed public exponent, RSA provides UEO. An intuitive argument runs thus: second key construction yields a key pair, from which a factorization of the modulus n' can be deduced. However, a pair (m', s) may accept only very few modulus values for which s will be a valid signature ($s^e \bmod n$ must be a valid padded value derived from m'). Hence, successful second key construction is a factorization algorithm on one of the modulus values n' for which $s^e \bmod n' = v'$. Easily factored integers of the size of a typical RSA modulus are scarce; hence, generic second key construction should not succeed in the general case.

Having a fixed public exponent means that the standard within which RSA is used specifies one unique public exponent which all public keys use. Although some values are popular (3, 17 and 65537), no clear consensus has yet been established on which value should always be used. Most protocols currently specify an RSA public key as a pair "modulus + public exponent", sometimes with restrictions on the public exponent size.

2.2 DSS

Algorithm Description

DSS is the American government Digital Signature Standard; it is described in [4]. The base algorithm uses computations in a subgroup of invertible numbers modulo p. The group parameters are:

- p, a 1024-bit prime number;
- q, a 160-bit prime number which divides $p - 1$;
- g, a multiplicative generator of a subgroup of order q (which means that $0 < g < p$, $g \neq 1$ and $g^q = 1 \bmod p$).

A DSS private key is an arbitrary number x such that $0 < x < q$, and the associated public key is $y = g^x \bmod p$.

In order to sign a message m, a hash function (SHA-1 for basic DSS) is applied to the message, yielding $H(m)$ (the size of $H(m)$ should be at least the size of q). Then the signer chooses a random number k such that $0 < k < q$ and computes the two following values:

$$\begin{aligned} r &= (g^k \bmod p) \bmod q \\ s &= k^{-1}(H(m) + xr) \bmod q \end{aligned}$$

The signature is the pair (r, s).

The signature verification algorithm is thus:

$$\begin{aligned} w &= s^{-1} \bmod q \\ u_1 &= wH(m) \bmod q \\ u_2 &= wr \bmod q \\ v &= (g^{u_1} y^{u_2} \bmod p) \bmod q \end{aligned}$$

The signature is accepted if and only if $v = r$.

Second Key Construction

We now consider that we have a valid triplet with a public key, a message and a signature, with the notations described above. We choose a new message m' and an arbitrary (e.g. random) number f such that $0 < f < q$. We now compute the following values:

$$\begin{aligned} g' &= (g^{u_1} y^{u_2})^f \bmod p \\ x' &= r^{-1}(sf^{-1} - H(m')) \bmod q \\ y' &= g'^{x'} \bmod p \end{aligned}$$

It can easily be seen that (r, s) then constitutes a valid signature for message m' relatively to the public key y', with group parameters p, q and g'. x', the private key corresponding to y', is known; thus, we have completed a successful second key construction.

Remarks

• The message m' in the algorithm described above is chosen arbitrarily and may be equal to the message m. Thus, this algorithm demonstrates that DSS has neither CEO nor DEO.

• The "trick" used here is to use g^k (as advertised by the initial signer through the signature) as the basis for a new group generator. DSS security relies on the fact that discrete logarithm is hard: retrieving k from $g^k \bmod p$ is not computationaly feasible. By shifting the generator, we make that discrete logarithm easy. We still use the same group. Since q is prime, and thus relatively prime to both k and f, a random choice of f yields just any group element with equal probability. This makes the second key exactly as valid as the initial key: no test on the public key or group parameters may establish which of the two keys is the "correct" one.

• It is not possible to perform second key construction without altering the paramaters p, q and g. If such a second key was found, the signature and the private key could be used to retrieve the original k parameter used by the original signer; the whole process would then solve discrete logarithm (find k from $g^k \bmod p$), which is considered intractable.

• DSS defines (in [4]) a parameter generation algorithm, which uses internal pseudo-random number generators. Once the seed is determined, the parameter generation algorithm is deterministic; hence, with the seed, the parameter generation algorithm can be run again and the parameters checked. This validation algorithm defeats the second key construction described above, because it is not possible to find a seed which will match the new group generator g' (that such a seed exists at all is actually most improbable). However, the publication of the seed is optional in the usual public key formats, the validation is very expensive (thousands of times the cost of verifying the signature, up to dozens of seconds on a big modern PC), and the standard parameter generation is suboptimal (it does not include some basic optimisations which may speed up the process). The bottom line is that some parameter generation algorithms do not use the standard, verifiable algorithm, and almost none of the verifiers do actually validate the parameters.

• The second key construction algorithm can trivially be applied to variants of DSS based on elliptic curves, or other groups. It also works with other schemes based on discrete logarithm, such as Schnorr[10] and El-Gamal[11]. The basic principle remains the same: shift the generator, using the published (either directly, or indirectly as in DSS) value g^k as basis for the new generator.

3 Adding Exclusive Ownership

In the following sections, we will concatenate and apply hash functions on both messages and public keys; this assumes that a convenient unambiguous deterministic binary representation of messages and public key is defined. This is in any way true for almost all existing applications, for instance those which handle digital certificates.

3.1 Destructive Exclusive Ownership

Destructive exclusive ownership can be added to any signature scheme in the following way: from a scheme (G, Σ, V), define a new scheme (G', Σ', V') such that:

- $G' = G$;
- for any message m, $\Sigma'(K_{\text{priv}}, m) = \Sigma(K_{\text{priv}}, m)||h(m)$ where $||$ is the concatenation, and h is a cryptographically secure hash function;
- for any signature $s = s_1||t$ where t has the size of the output of h, s is accepted by V' if and only if s_1 is accepted by V and $t = h(m)$.

If the hash function h is collision-free (it is computationaly infeasible to exhibit a collision), that new scheme trivially provides DEO, because the signature is “branded” with the signed message, and hence cannot be used for any other message. Note that if the original scheme does not provide CEO, the new scheme does not provide it either.

3.2 Conservative and Universal Exclusive Ownership

Conservative exclusive ownership can be added to any signature scheme in the following way: from a scheme (G, Σ, V), define a new scheme (G', Σ', V') such that:

- $G' = G$;
- for any message m, $\Sigma'(K_{\text{priv}}, m) = \Sigma(K_{\text{priv}}, m)||h(K_{\text{pub}})$ where $||$ is the concatenation, and h is a cryptographically secure hash function;
- for any signature $s = s_1||t$ where t has the size of the output of h, s is accepted by V' if and only if s_1 is accepted by V and $t = h(K_{\text{pub}})$.

If the hash function h is collision-free, this scheme makes each signature exclusive to the public key which was used to generate it. Thus, the signature cannot be reused with any other public key. It so happens that this provides UEO, which is the union of CEO and DEO.

It is possible to create a contrived example which ensures CEO but *not* DEO: for instance, use DSS and define the signature to be the concatenation of a plain DSS signature on the message, and $h(m \oplus (p, q, g))$ where $\oplus$ is a bitwise exclusive or, and p, q and g are the public key parameters (some formatting and padding is needed in order to define this cleanly). Since second key construction on DSS requires changing the parameters (see section 2.2), m must also be modified in order to keep the total hash value unmodified; but such modification is easy since a simple bitwise exclusive or is used. Hence this scheme does not provide DEO. But it can be proven that if this modified DSS does not provide CEO, then the attacker can also produce existential forgeries on DSS.

This example shows that CEO does not imply UEO.

3.3 UEO Without Expanding Signatures

The method described in the previous section provides UEO but increases the signature size, which may be inappropriate for some applications. Unfortunately, we found no easy way to define a generic construction which provides UEO without increasing signature size. However, we do have a construction which works if the underlying signature scheme provides an additional property, which we explicit below.

Let us consider a secure signature scheme (G, Σ, V) which has the following property $\mathcal{P}$: for any given public key K_{pub} and signature value s that successfully pass the basic correctness tests that the verifier is willing to implement, and for random messages m taken in a defined space $\mathcal{H}$ (of cardinal $\#\mathcal{H}$), the probability over $\mathcal{H}$ that $V(K_{\text{pub}}, m, s) = \textbf{true}$ is negligeable (no more than 2^{-80} for instance). For the purpose of our construction, it is not needed that this property be true for any space $\mathcal{H}$, but only for the space of output values of a proper hash function h.

RSA has that property. The signature s and the public key (n, e) are sufficient to compute $s^e \bmod n$, which is the padded message, which is mostly the hashed message with additional features. For DSS, however, the situation is a bit more problematic: if the verifier does not perform some validation on the public key parameters, this property is not met (for a given signature, it is possible to devise a flawed public key such that the signature will be considered valid for many possible messages, for instance one message in three). The validation process must check that:

- q is prime;
- r and s (the two parts of the DSS signature) are strictly lower than q;
- g (the group generator) is greater than 1, lower than p, and has order q;
- y (the public key itself) is greater than 1, lower than p, and has order q.

Under those conditions, DSS has the required property $\mathcal{P}$. Performing these checks will slow down signature verification by a factor of about 2.5 in a typical implementation.

We will now build the new signature scheme (G', Σ', V') which provides UEO. We suppose that there exist:

- a secure hash function h which takes as input arbitrary bit strings;
- an unambiguous deterministic representation of the possible output values of h into messages m in $\mathcal{M}$.

Then we define:

- $G' = G$;
- $\Sigma'(K_{\text{priv}}, m) = \Sigma(K_{\text{priv}}, h(m||K_{\text{pub}}))$ (where $||$ denotes concatenation);
- $V'(K_{\text{pub}}, m, s) = V(K_{\text{pub}}, h(m||K_{\text{pub}}), s)$.

We place ourself in the (non-standard) model known as the Random Oracle Model, proposed in [12] after a suggestion by Fiat and Shamir[13]. If we consider h to be a random oracle, and if the original signature scheme has the property defined above, then this new signature scheme provides UEO. This is easily proven: the very nature of h as a random oracle means that second key construction has to work as a guess-work, where both m' and K'_{pub} are chosen prior to invoking the oracle, with a probability of success which is negligeable, thanks to the property of the initial signature scheme. And since K'_{pub} is distinct from K_{pub} (by definition of second key construction), none of those requests to the oracle may be performed by the initial user; thus, messages signed by the initial user are of no help to the attacker.

In most usual signature schemes, an appropriate hash function h' is already used, which means that the signature scheme operates on $h'(m)$ instead of m directly. We can thus merge that internal hash function h' with h. In that sense, we can give UEO to such a signature protocol by simply adding a copy of the public key (or a hash thereof) within the signed data. Note that, for any signature scheme which is deemed resistant to existential forgeries, signing $m||K_{\text{pub}}$ is *not* sufficient to guarantee UEO, unless resistance to existential forgeries is true for *all* possible keys, even the weak and potentially incorrect keys that the attacker may use. Hence the property $\mathcal{P}$ which we defined above.

4 Conclusion

We presented in this article the new notion of Exclusive Ownership, which usual signature algorithms do not provide. That notion is important to some real-world applications, such as the X.509 revocation system through CRLs. We showed how to defeat EO for some algorithms such as RSA and DSS; we also presented some generic ways to modify signature schemes in order to provide UEO, although these methods have shortcomings (either signature size increase, or restriction to algorithms which have a specific property, and the security proofs are in the random oracle model).

Our model describes the attack as recontructing a new key pair, which the attacker may later use to sign other messages. A slightly weaker attack model would be the following: the attacker builds a new public key K'_{pub}, which validates a previously issued signature $s' = s$ on some message m', and also a new signature s'' on another message m'', distinct from m'. In this model, the attacker needs not gain any knowledge of a private key K'_{priv} matching K'_{pub}. Our generic constructions for providing exclusive ownership also defeat this weaker attack model. It is yet an open question, whether RSA with a fixed or range-restricted public exponent provides exclusive ownership, either in our main attack model, or in this weaker model.

References

1. *Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile*, R. Housley, W. Polk, W. Ford and D. Solo, RFC 3280, April 2002.

2. *On the Soundness of Certificate Validation in X.509 and PKIX*, T. Pornin and J. P. Stern, to appear in EuroPKI 2005.
3. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, J. Jonsson and B. Kaliski, RFC 3447, February 2003.
4. *Digital Signature Standard*, National Institute of Standards and Technology (NIST), FIPS 186-2, 2000.
5. *Key-spoofing attacks on nested signature blocks*, R. Christianson and M. R. Low, Electronics Letters, vol. 31, no. 13, 1995, pp. 1043–1044.
6. *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*, S. Goldwasser, S. Micali and R. Rivest, SIAM Journal on Computing, vol. 17, no. 2, 1988, pp. 281–308.
7. *Flaws in Applying Proof Methodologies to Signature Schemes*, J. Stern, D. Pointcheval, J. Lee and N. Smart, Lecture Notes in Computer Science, Proceedings of Crypto'02, 2002, pp. 93–110.
8. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, R. Rivest, A. Shamir and L. Adleman, Communications of the ACM, February 1978, pp. 120–126.
9. *Fast decipherment algorithm for RSA public-key cryptosystem*, J.-J. Quisquater and C. Couvreur, Electronics Letters, vol. 18, no. 21, October 1982, pp. 905–907.
10. *Efficient signature generation by smart cards*, G.P. Schnorr, Journal of Cryptology, vol. 4, 1991, pp. 161–174.
11. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, T. El-Gamal, Lecture Notes in Computer Science, Proceedings of Crypto'84, 1985, pp. 10–18.
12. *Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols*, M. Bellare and P. Rogaway, Proceedings of the 1st CCS, ACM Press, 1993, pp. 62–73.
13. *How to Prove Yourself: Practical Solutions of Identification and Signature Problems*, A. Fiat and A. Shamir, Lecture Notes in Computer Science, Proceedings of Crypto'86, 1987, pp. 186–194.