Timed Release of Standard Digital Signatures

[Extended Abstract]

Juan A. Garay[1] and Markus Jakobsson[2]

[1] Bell Labs – Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974.
E-mail: garay@research.bell-labs.com. URL: www.bell-labs.com/user/garay.
[2] RSA Labs, Bedford, MA 01730. URL: www.markus-jakobsson.com.

Abstract. In this paper we investigate the timed release of *standard* digital signatures, and demonstrate how to do it for RSA, Schnorr and DSA signatures. Such signatures, once released, cannot be distinguished from signatures of the same type obtained without a timed release, making it transparent to an observer of the end result. While previous work has allowed timed release of signatures, these have not been standard, but special-purpose signatures.
Building on the recent work by Boneh and Naor on timed commitments, we introduce the notion of a *reusable time-line*, which, besides allowing the release of standard signatures, lowers the session costs of existing timed applications.

1 Introduction

While probably the most important pillar of cryptography is the believed existence of hard problems, the notion of *moderately hard* problems also bears promise. A moderately hard problem is one which is not computationally infeasible to solve, but also not easy. Since Dwork and Naor introduced the notion for purposes of spam protection [DN92], considerable research has gone into the area, and the methods have been refined. A first type of work relating to timing mechanisms is that of proposing and developing suitable primitives, for which good lower bounds can be developed. Another avenue of research relates to applications that *use* timed mechanisms. This paper makes progress in both of these areas by extending the recent work of Boneh and Naor on timed commitments and timed signatures [BN00] to (1) allow for the reuse of their proposed timed structure, thus achieving lower session costs, and (2) allow for the robust timed release of standard signatures.

Background and motivation. We will follow the convention and refer to schemes with a moderately hard computational "trigger" as *timed*, since the effort involved relates closely to the time of performing the computation, and, functionally speaking, it is often the time rather than the effort that is relevant. To this end, primitives have been proposed that do not allow for easy parallelization of the problem solving. This shifts the emphasis from a *global* computational effort to the local effort, which is more closely related to actual time than to CPU time. For this purpose, problems based on modular exponentiation are believed to be good, as considerable effort has been invested in finding efficient exponentiation algorithms, and it is believed to be a problem not well suited for parallelization. For this reason, and following [RSW96,BN00], we base our work on iterated squaring.

While the computational lower bounds for the *problem solver* are paramount, and have been the focus of previous research, the computational costs of the *problem generator* (resp., the *solution verifier*) have not received the same attention. One focus of our paper is to curb the computational costs to generate and verify problems, while maintaining other desirable properties (such as protection against parallel attacks). Of these two costs, it is the problem generation cost that is the most difficult to reduce, and that which we concentrate on. We introduce the notion of *reusable time-lines*, which allow an amortization of the generation costs. With this benefit, we also get a reduced communication complexity, as large portions of the interaction and verification can be performed at setup, rather than at each session.

The resulting primitive can be used to reach a new goal: the timed release of *standard* digital signatures. While previous work has allowed timed release of signatures [Sha84,FS86,BN00], these have not been standard signatures, but special-purpose signatures. The robustness properties of our reusable time-lines allow us to get timed release of standard signatures; we show how to do it for RSA, DSS and Schnorr signatures. Such signatures, once released, cannot be distinguished from signatures of the same type obtained without a timed release. This makes the timed release transparent to an observer of the end result, and allows for an integration of timed release in legacy systems.

Our methods allow for easy integration with fair exchange methods, and they can also be used to obtain timed escrowing of ciphertexts (whether in conjunction with additional decryption keys or without). Such releases are efficient if certain encryption schemes (e.g., ElGamal) are employed. In this extended abstract we will concentrate on the signatures application.

Prior and related work. We already mentioned the work of Dwork and Naor [DN92] on moderately hard functions, which they use to combat junk mail.[1] The idea of "sending information into the future" is attributed to May [May93], who discussed several applications, mainly in the context of encrypting information.

Timed primitives have been designed for cryptographic key escrow. In [Sha95], Shamir suggested to escrow only partially a DES key, so that to recover the whole key, the government would need to first obtain the escrowed bits of the key, and then search exhaustively for the remaining bits. A problem with this type of approach is that the search can be parallelized, thus making it difficult to give a precise bound on the number of steps needed for the recovery. Bellare and

[1] In this extended abstract, we only review time-related work that is more closely related to ours.

Goldwasser [BG96,BG97] later suggested the notion of "time capsules" for key escrowing in order to deter widespread wiretapping, and where a major issue is the verification at escrow time that the right key will be recovered.

In another line of work, Rivest, Shamir and Wagner [RSW96] suggested "time-lock puzzles" for encrypting data, where the goal is to design puzzles that are "intrinsically sequential," and thus, putting computers to work together in parallel does not speed up finding the solution. They base their construction on a problem that seems to satisfy that: modular exponentiation. In their work, however, no measures are taken to verify that the puzzle can be unlocked in the desired time.

Using a function similar to that of [RSW96], Boneh and Naor [BN00] suggest the notion of (verifiable) *timed commitments*, an extension to the standard notion of commitments in which a potential forced opening phase permits the receiver to recover (with effort) the committed value vithout the help of the committer. They show how to use timed commitments to improve a variety of applications involving time, including timed signatures, contract signing, honesty-preserving auctions, and concurrent zero-knowledge. Their notion and construction of timed commitments is the starting point for our work.

Our work. More specifically, we build on the Boneh-Naor structure for timed commitments. We call this structure a "time-line." In a nutshell, a time-line is a vector of elements, where each element is obtained by iterated squaring of the previous element. The time-line is represented by its endpoints, and the commitment it corresponds to is a value on the line. This value can be computed by iterated squaring of one of the values representing the time-line.[2]

A first contribution of our paper is the notion of *derived* time-lines, which are time-lines that are obtained by "shifting" of a master time-line. The benefit of doing this is that while expensive proofs are needed to verify the correctness of time-lines in [BN00] for *each* commitment, we only have to perform this computation once (during a setup phase), after which a much simpler (and less expensive) verification of correct shifting can be used. Derived time-lines are "backwards compatible," in the sense that they can be readily used to solve the same problems addressed in [BN00].

Moreover, our construction allows for another application. The second – and main – contribution of our work is the use of our derived time-lines (and the corresponding commitments) for a robust release of signatures. While the signatures released in [BN00] are "special-purpose" signatures (designed with a timed release in mind), we can release *standard* signatures. Essentially, we use the committed value from the time-line as a "blinding" factor for the signature [Cha82]. However, providing the "connective tissue" between the time-line component and the signature component of the timed signature protocol is an important element of our construction. We demonstrate this by exhibiting a timed release

[2] In [BN00] this vector is used to perform timed commitments to arbitrary strings. Thus, what we call a (commitment to a) time-line is a simpler building block: given the time-line the committed value is fixed; we elaborate on this onwards.

of RSA, Schnorr and DSA signatures. For RSA signatures, we require for simplicity that the party releasing the signature (the committer) has generated the signature, while the release protocols for Schnorr and DSA signatures allow the release of signatures generated by third parties; we show how to do the third party release for RSA in the full version of the paper. Allowing the timed release of signatures by parties other than the signer may be a beneficial property in payment schemes, among other applications.

Organization of the paper. The cryptographic tools and assumptions that are used in our constructions are given in Section 2. The notion of a time-line and the protocol for new time-line generation and commitment are presented in Section 3. The definition of timed release of a standard signature, together with the protocols for timed RSA, Schnorr and DSA signatures, are given in Section 4.

2 Cryptographic Tools and Assumptions

Let N be a Blum integer, i.e., $N = pq$, where p and q are distinct primes each congruent to 3 mod 4. Recall the notion of a Blum-Blum-Shub (BBS) sequence $x_0, x_1, \cdots, x_n$, with $x_0 = g^2 \pmod N$ for a random $g \in \mathbb{Z}_N$, and $x_i = {x_{i-1}}^2 \pmod N$, $1 \leq i \leq n$. It is shown in [BBS86] that the sequence defined by taking the least significant bit of the elements above is polynomial-time unpredictable (unpredictable to the left and to the right), provided the quadratic residuosity assumption (QRA) holds.

The generalized BBS assumption. In [BN00], Boneh and Naor postulate the following generalization of unpredictability of BBS-type sequences. Let N and g be as above, and k an integer such that $l < k < u$. Then given the vector

$$< g^2, g^4, g^{16}, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^{k-1}}}, g^{2^{2^k}} > (\mathrm{mod} N), \tag{1}$$

the (l, u, δ, ϵ) generalized BBS assumption states that no PRAM algorithm whose running time is less than $\delta \cdot 2^k$ can distinguish the element $g^{2^{2^{k+1}}}$ from a random quadratic residue R^2, with probability larger than ϵ. The bound l precludes the parallelization of the computation, while the bound u precludes the feasibility of computing square roots through factoring. We refer to [BN00] for further details.

Equality of discrete logs. Our constructions will be using (statistical) proofs of knowledge (PK) of equality of two discrete logs modulo the same number (say, n), or in *different* moduli. Proofs for equality modulo the same number have been considered in [CEvdG87,CP92,Bao98]. We will use EQLOG-1(x, n) to refer to these proofs.

Proofs for equality of two (or more) discrete logs (alternatively, a discrete log and a committed number, or two committed numbers) in different moduli have been considered in [BT99,CM99]. Specifically, and following [BT99], let t, l and

s be three security parameters, and let n_1 be a large composite number whose factorization is unknown to the prover, and n_2 be another large number, prime or composite whose factorization is known or unknown to the prover. Let g_1 be an element of large order of $\mathbb{Z}_{n_1}^*$ and h_1 be an element of the group generated by g_1 such that both the discrete logarithm of g_1 in base h_1 and the discrete logarithm of h_1 in base g_1 are unknown to the prover. Let g_2 be an element of large order of $\mathbb{Z}_{n_2}^*$. Assume that the prover knows an integer $x \in \{0, \ldots, b\}$. It is shown in [BT99,CM99] how to give an efficient (non-interactive) statistical proof of knowlege

$$\mathrm{PK}(x, r_1 : y_1 = g_1^x h_1^{r_1} \bmod n_1 \wedge y_2 = g_2^x \bmod n_2),$$

where r_1 is randomly selected from $\{-2^s n_1 + 1, \ldots, 2^s n_1 - 1\}$, and where completeness holds with probability at least $1 - 2^{-l}$ and soundness holds with probability at least $1 - 2^{-(t-1)}$. We will use EQLOG-2(x, n_1, n_2) as a short form for the above proof. The role of n_1, whose factorization is unknown to the prover, is that of a "helper" modulus, and can be constructed as in [CM99]. In our constructions we will omit a reference to the helper modulus for simplicity.

Regarding the size of x, however, what is proven is that $x \in [-2^{t+l}b, 2^{t+l}b]$ – i.e., an $O(2^{t+l})$ *expansion rate*. In our applications, we will have the additional requirement that the discrete log lie in a specific bounded range, which leads us to the next building block.

Proof that a discrete log lies in an interval. The idea of checking whether a committed integer lies in a specific interval has many cryptographic applications. It was first developed in [BCDvdG87], and has been later studied in [CFT98,CM99,Mao98,Bou00]. In [Bou00], Boudot presents an efficient method to prove that a committed number $x \in I$ belongs to I, and not to a larger interval (i.e., an expansion rate of 1).

Specifically, let t, l and s be three security parameters, and let n_1 be a large composite number whose factorization is unknown to the prover, as above. Let integer $x \in [a, b]$ and $y_1 = {g_1}^x {h_1}^{r_1} \bmod n_1$, where r_1 is randomly selected from $\{-2^s n_1 + 1, \ldots, 2^s n_1 - 1\}$. It is shown in [Bou00] how to give an efficient (non-interactive) statistical proof of knowlege

$$\mathrm{PK}(x, r_1 : y_1 = {g_1}^x {h_1}^{r_1} \bmod n_1 \wedge x \in [a, b]).$$

Combined with the proof of knowledge of discrete logs in different moduli mentioned before, the above protocol can be used to prove that a discrete logarithm modulo p (a prime number or a composite number whose factorization is known to the prover) belongs to an interval. Refer to [Bou00] for details. We use INTVL$(x \in [a, b])$ as a short form for the above proof.

3 Time-Lines

In this section we define the notions of a *time-line* and the *commitment* to a time-line. In a nutshell, a time-line is the partial description of a BBS sequence, as given by the input vector (1) in the generalized BBS assumption. In [BN00] this vector is used to perform *timed commitments*[3] to arbitrary strings. The simpler building block of a time-line commitment (defined below) will be useful for this paper's main result, namely, the timed release of standard signatures (Section 4); it will also allow us to make timed commitments (as well as the other applications in [BN00]) more efficient (Section 3.2).

Let N be a Blum integer and g an element of large odd order in $\mathbb{Z}_N^*$ (see [BN00] for ways for choosing g so as to guarantee this). A *value-hiding time-line* (*time-line* for short) is a partial description of a BBS sequence, given by the subsequence $\{g^{2^{2^i}}\}_{i=0}^{k}$ (mod N). We call the value $g^{2^{2^k-1}}$ the time-line's *hidden value*.[4] Informally, we say that a value *lies on* a time-line if it can be obtained by iterated squaring of the time-line's start value.

Now the notion of a (T, t, ϵ) *time-line commitment* naturally follows. It enables the committer to give the receiver a timed commitment to a hidden value. At a later time, she can reveal this value and prove that it's the correct one. However, in case the committer fails to reveal it, the receiver can spend time T and retrieve it.

As in the case of the timed commitment of [BN00], a (T, t, ϵ) time-line commitment consists of three phases: the **commit** phase, where the committer commits to the time-line's hidden value by giving the receiver the description of the time-line and a proof of its well-formedness; the **open** phase, where the committer reveals information to the receiver that allows him to compute the hidden value — and be convinced that this is indeed the committed value; and the **forced open** phase, which takes place when the committer refuses to execute the open phase; in this case, the receiver executes an algorithm that allows him to retrieve the hidden value, and produces a proof that this is indeed the value.

A time-line commitment scheme must satisfy the following security constraints:

Binding: The value committed to is uniquely defined by the commitment. In particular, it is not possible to open up the commitment to a value other than that which will be obtained by the forced opening of the commitment (corresponding to the iterated squaring of the time-line's starting value.)

Soundness: After receiving the time-line, the receiver is convinced that the forced open phase will produce the hidden value in time T.

Privacy: Every PRAM algorithm whose running time is at most $t < T$ on polynomially many processors, given the transcript of the time-line commit protocol as input, will succeed in computing the time-line's hidden value with probability at most ϵ.

Proving the well-formedness of a time-line, as described below, involves a computational effort. However, once a time-line is established and its proper-

[3] A timed commitment is an extension to the standard notion of commitments in which a potential forced opening phase permits the receiver to recover (with effort) the committed value vithout the help of the committer [BN00].

[4] The notion easily extends to more (polynomially many) hidden values. For simplicity, we just consider one, as this will be enough for our main application.

ties verified (call it the *master* time-line), we can generate subsequent time-line commitments at a low cost. Intuitively, the idea is for the committer to create a new time-line from the master time-line by applying a secret transformation value – the *shifting* factor – to the end points of the master time-line, in such a way that verification of the new time-line's properties is easy. We now present the protocols for time-line commitment in more detail.

3.1 Our time-line commitment protocol

During the commitment setup phase below, the master time-line is established and its well-formedness proven. The setup process follows the commitment protocol of [BN00] almost *verbatim*, except for modifications that are needed for the subsequent, and derived, time-line commitments. (These modifications make it possible to use (and re-use) the time-lines for transparent release of standard signatures, as will be shown later.)

Setup. Let $T = 2^k$ be an integer, and N as defined above, i.e., $N = pq$ for p, q two random n-bit primes such that $p = q = 3 \bmod 4$. We assume N is a publicly known modulus, associated either with time-line commitments in general, or with time-line commitments from one particular user. The generator g of the master time-line is chosen so that the largest prime divisor of the order of g in $\mathbb{Z}_N^*$ is large. Specifically, in [BN00] the party performing the setup picks a random $f \in \mathbb{Z}_N$ and computes $g = f^{\left(\prod_{i=1}^{p} q_i^n\right)} \bmod N$, where $q_1, q_2, \cdots, q_p$ are all the primes less than some bound B. Upon receiving f and g the receiver is assured that the order of g in $\mathbb{Z}_N^*$ is not divisible by any primes smaller than B. Next, the following steps are performed by the party generating the master time-line.

1. *Compute master time-line.* Compute $M_i = g^{2^{2^i}} \bmod N$, $0 \le i \le k$, $m_{1\mathrm{st}} = g^{2^{2^k-1}} \bmod N$, and $m_{2\mathrm{nd}} = g^{2^{2^k-2}} \bmod N$.

   If the party knows $\varphi(N)$ (as in the case where a trusted party generates the master time-line), this computation can be performed by computing $a_i = 2^{2^i} \bmod \varphi(N)$ and then $M_i = g^{a_i} \bmod N$. If $\varphi(N)$ is unknown to the party, then it can be performed by iterated squaring.

2. *Prove well-formedness.* Generate a proof that $M_i = g^{2^{2^i}} \bmod N$, for $0 \le i \le k$. This is done by showing that each triple $< g, M_i, M_{i+1} >$, $0 \le i < k-1$, is of the form $< g, g^x, g^{x^2} >$, for some x.[5]

3. *Output master time-line.* Output $\{M_i\}_{i=0}^k$, $m_{1\mathrm{st}}$ and $m_{2\mathrm{nd}}$, along with the above proof of well-formedness, which we assume is non-interactive. Before using the master time-line, any party checks that ${m_{2\mathrm{nd}}}^2 = m_{1\mathrm{st}} \pmod N$, ${m_{1\mathrm{st}}}^2 = M_k \pmod N$, and that the proof is correct.

[5] These k proofs are based on the classical zero-knowledge proof that a tuple is a Diffie-Hellman tuple, take fours rounds, and can be performed in parallel – see [BN00] for details. The requirement that the smallest prime divisor of g's order is large is needed for the soundness of these zero-knowledge proofs [CP92]; see also [GMP01].

We now show how to generate a new time-line based on the master time-line.

Commit phase. The committer performs the following steps:

1. *Choose shifting factor.* The committer picks at random a value $\alpha \in \mathbb{Z}_{\tilde{\varphi}}$, where $\tilde{\varphi} = \varphi(N)$ is used if known, and otherwise $\tilde{\varphi} = \lfloor N - 2\sqrt{N} \rfloor$.
2. *Compute new time-line.* The committer computes $h = g^\alpha$, $R_{k-1} = (M_{k-1})^\alpha$, $r_{\text{aux}} = (m_{\text{2nd}})^\alpha$, $r = (m_{\text{1st}})^\alpha$, and $R_k = (M_k)^\alpha \,(\operatorname{mod} N)$. He outputs h, R_{k-1} and R_k, and **keeps** r **and** r_{aux} **secret**. r is the new time-line's hidden value. (r_{aux} will be used in our timed signature application of Section 4.)
3. *Prove well-formedness of new time-line.* The committer proves to the receiver that
$$\log_g h = \log_{M_{k-1}} R_{k-1} = \log_{M_k} R_k \, (= \alpha),$$
i.e., that the new time-line is correctly derived from the master time-line. He uses EQLOG-1(α, N) for this proof.

Sometimes we will use TL_α to refer to the time-line generated from the master time-line by applying α, and $\mathrm{TL}_\alpha(r)$ to refer to the commitment to the time-line's hidden value. We will drop the subscript when clear from the context.

Open phase. The committer sends α and $f' = f^{2^{(2^k-1)}}$ to the verifier, who computes $r = (f'^\alpha)^{\prod_{i=1}^{p} q_i^n} \bmod N$, and checks that $h = g^\alpha \bmod N$ and $r^2 = R_k \bmod N$. At this point the receiver has a square root of R_k. Having odd order ensures that r is in the subgroup generated by h.

Forced open phase. The user computes r from R_{k-1} by repeated squaring mod N; specifically, $2^k - 2$ operations.

Theorem 1. *Assume the hardness of the discrete logarithm problem and that the (l, u, δ, ϵ) generalized BBS assumption holds. Then, for $k > l$, the scheme above is a secure (T, t, ϵ) time-line commitment scheme with $t = \delta \cdot 2^k$ and $T = M(u) \cdot 2^k$, where $M(u)$ is the time it takes to square modulo an u-bit number.*

Proof (sketch). Note that, by construction, the order of h does not have any small prime divisors. Thus, the EQLOG-1 proofs in step 3 of the commit phase guarantee that h, R_{k-1} and R_k lie on the new time-line. This gives soundness since it proves that the beginnings of the time-lines (g, resp., h) are related to each other like the ends of the time-lines ($g^{2^{2^k}}$, resp., R_k). Thus, if the master time-line is correct, so is the derived time-line, since it establishes that $R_k = h^{2^{2^k}}$.

For the binding property, during the open phase, the receiver receives f' which leads to r satisfying $r^2 = R_k \bmod N$. Furthermore, r has odd order in $\mathbb{Z}_N^*$. During the commit phase, the verifier is also assured that h has odd order in $\mathbb{Z}_N^*$, and that R_k is in the subgroup generated by h (soundness property). The claim then follows since R_k has a unique square root in the subgroup (odd order), and in $\mathbb{Z}_N^*$ there can be at most one square root of R_k of odd order.

For privacy, the generalized BBS assumption protects from computing r from R_{k-1} in a number of operations significantly less than 2^k squarings, and the difficulty of factoring from computing it from R_k; the hardness of the discrete logarithm problem prevents any information about α to be leaked. ■

3.2 Efficiency and applications of time-line commitments

In [BN00], Boneh and Naor show how time-lines can be used for a variety of applications involving time, including timed commitments, timed (non-standard) signatures, contract signing, collective coin flipping, honesty-preserving auctions, and concurrent zero-knowledge. Specifically, to perform a timed commitment to a message m of length ℓ (assume for simplicity that $\ell \leq |N|$), the committer hides the message by computing $c_i = m_i \oplus g^{2^{2^k - i}}$, $1 \leq i \leq \ell$, and sends c to the receiver together with the time-line parameters.

The timed signature scheme of [BN00] is obtained from the timed commitment scheme above as follows. To obtain a timed signature on a message m, the signer first performs a timed commitment to a random secret string m', thus obtaining c', and then signs the pair (m, c') – call this signature $\mathrm{Sig}(m, c')$. Upon retrieving m' from the commitment's open (or forced open) phase, the verifier submits the triple $< m', c', \mathrm{Sig}(m, c') >$ as the proof of a valid signature on m.

Note that our time-line commitments (alternatively, derived time-lines) described above can be used in the same way to obtain timed commitments to arbitrary strings (resp., timed signatures), at the expense of an additional hardness assumption. The efficiency gains, however, will be substantial, as we would essentially perform the large portion of the work only once, in the setup session, and then amortize the cost of this over many re-uses of the master time-line. More specifically, the Boneh-Naor scheme – and our setup session – requires the proof (and verification) of the "B-roughness" of g, the generation of the time-line vector, and $O(k)$ proofs of equality of discrete logs for its well-formedness. However, once the setup is performed and the master time-line established, we only need to perform a constant number of exponentiations (3) for the generation of the new time-line, and a number of equality of discrete log proofs as required by the timed application. For example, in collective (two-party) coin flipping, where party A time-commits to a bit b_A to party B, and B just sends his bit b_B to A, with the collective coin being $b_A \oplus b_B$ (see [BN00]), only 2 such proofs are needed. In the timed commitment and timed signature scheme above, $O(\ell)$ proofs would be required. The open phases, whether forced or standard, have very similar complexity in the two schemes.

In the next section, we give another important application of time-line commitments: how to generate timed signatures in a "transparent" way, i.e., with no modifications to the arguments or definition of a valid signature.

4 Timed Release of Standard Signatures

Our approach to allow for the timed release of standard signatures (or other cryptographic function) is to use the hidden value from the time-line commitment, r, as a *blinding* factor [Cha82] for the signature. However, in order to obtain robustness for this operation, one has to prove that r is uniquely defined, that is, obtainable from both forced and standard decommitment, and correctly applied to blind the signature (or other function). Secondly, the prover has to perform a signature-dependent proof that r is the blinding value used to derive the blinded signature given to the receiver, and thus, that the receiver will be able to obtain the unblinded signature once he has computed or obtained the blinding factor r. Since operations will be taking place in different moduli, care has to be taken here to rule out the possibility that a value congruent to r, but not equal to r, is used for the blinding. Before showing how these objectives are achieved, we give a more formal definition of our timed signature scheme.

Given a regular signature scheme that allows for blinding (e.g., RSA, Schnorr, DSA), let $\mathrm{Sig}(m)$ denote the signature on message m, generated with the signer's private key, and which verifies with the signer's public key. Our (T, t, ϵ) timed signature scheme will consist of the following phases:

Commit phase: The signer performs a (T, t, ϵ) time-line commitment, and uses the time-line's hidden value r to blind the signature — call it $\overline{\mathrm{Sig}}_r(m)$. The signer gives the pair $< \mathrm{TL}(r), \overline{\mathrm{Sig}}_r(m) >$ to the verifier, together with the proofs of well-formedness of the time-line, uniqueness of the blinding factor, and correctness of the blinding operation.

Open phase: The signer reveals r by opening the time-line commitment. The verifier uses r to unblind the signature and obtain $\mathrm{Sig}(m)$.

Forced open: The verifier uses the forced open algorithm from the time-line commitment to retrieve r, and uses it to unblind the signature as above.

Note on blinding values. In the following, we describe the blinding and recovery of RSA, Schnorr and DSA signatures. For all of these, we note that the blinding factor r that the verifier obtains from the time-line commitment is applied "as is," i.e., without any modification to make it an exponent of the right length; in particular, the exponent may be much larger than the order of the group. Similarly, in the preparation of the blinded signature, the blinding and all proofs of correct blinding are performed with respect to this unreduced value. While the prover may in some instances use a number not equal to r, but merely congruent, the structure of our proofs guarantee that the blinding and unblinding portions cancel out. Thus, the resulting unblinded signature is the expected and valid signature committed to. The fact that r is potentially much larger than a normal blinding factor for the particular signature scheme only amounts to some extra work in the blinding and the unblinding, and a skewing of the probability distribution of blinding factors [6]

[6] Skews in the probability distributions of exponents have recently been shown to be problematic, but only in situations with a very large reuse of the same exponent (such

4.1 Timed RSA signatures

Before we recall the RSA parameters, we introduce some additional elements that will be used in the proof of uniqueness of the blinding factor. (These will be common to the other signature schemes.) Let N be the publicly known modulus associated with time-line commitments from Section 3.1. We will be performing auxiliary (standard) commitments using a subgroup of order N in $\mathbb{Z}^*_{N'}$, for $N' = \kappa N + 1$ a prime; let $g\prime$ be a generator of this subgroup. These commitments will allow us to establish the desired relation between the time-line's hidden value and the blinding factor.

Now let n be an RSA modulus, (e, n) be the signer's public key, and d his secret key, chosen in such that way that $m^{ed} = m \bmod n$ for all values $m \in \mathbb{Z}_n$. The signer's signature on a (hashed) message m is therefore $s = m^d \bmod n$.

Commit phase. We assume for simplicity that the signer and the verifier have already done a time-line commitment $\mathrm{TL}(r)$. The signer performs the following steps:

1. *Normal signature generation.* Let m be the message to be signed. The signer computes $s = m^d \bmod n$.
2. *Application of blinding factor.* The signer blinds the signature by computing $\tilde{s} = s^{1/r} \bmod n$, where r is the time-line's hidden value, and sends the pair $(m, \tilde{s})$ to the verifier.
3. *Auxiliary commitments and proof of uniqueness.* The signer computes

$$b_{\mathrm{aux}} = g\prime^{r_{\mathrm{aux}}}, \quad b = g\prime^{r}, \text{ and } \; B = g\prime^{R_k} \pmod{N'}.$$

The signer then proves that $\log_{g\prime} b_{\mathrm{aux}} = \log_{b_{\mathrm{aux}}} b \,(= r_{\mathrm{aux}})$, using EQLOG-1-$(r_{\mathrm{aux}}, N')$; that $\log_{g\prime} b = \log_b B \,(= r)$, using EQLOG-1$(r, N')$; and that r and r_{aux} lie in the right interval using INTVL$(r \in [0, N-1])$ and INTVL$(r_{\mathrm{aux}} \in [0, N-1])$, respectively.[7]

4. *Proof of correct blinding.* The verifier computes $X = \tilde{s}^e$. The signer proves that $\log_X m = \log_{g\prime} b \,(= r)$, using EQLOG-2$(r, n, N')$.

Open and forced open phases. The verifier obtains r from the signer or from the time-line forced open algorithm. He then unblinds the signature by computing $s = \tilde{s}^r \bmod n$.

Theorem 2. *Under the hardness of the discrete logarithm problem and the generalized BBS assumption, the scheme above is a (T, t, ϵ) timed RSA signature scheme.*

as repetitive signing) [Ble00]. We note that each blinding factor is used only once, and on signatures whose distribution is (assumed to be) correct, and not affected by the blinding. Therefore, the skewed distribution does not represent a security concern here.

[7] Using the techniques of [CDS94], some of these proofs can be combined. Here we present them separately for clarity.

Proof (sketch). We first prove that the scheme is "binding," i.e., that after successful completion of the commit phase, the receiver is convinced that he will be able to retrieve the correct signature. This will follow from the fact that our time-line commitment is binding (Theorem 1), a proof of uniqueness (the same value committed to is used for the signature blinding), and a proof of correct blinding (once retrieved, the value committed to will produce the correct signature).

To see uniqueness, we first establish that the value from the auxiliary commitment, call it r', is the same as the value r hidden by the time-line (step 3 of the commit phase). Note that the proofs that $\log_{g'} b = \log_b B$ and INTVL($r \in [0, N-1]$) establish that $R_k = {r'}^2 \bmod N$, i.e., that r' is the square root of the value $R_k = h^{2^{2^k}} \bmod N$ that the verifier knows. Similarly, the proofs that $\log_{g'} b_{\text{aux}} = \log_{b_{\text{aux}}} b$ and INTVL($r_{\text{aux}} \in [0, N-1]$) establish that $r' = {r_{\text{aux}}}^2 \bmod N$, and hence that $r' \in Q_N$ (the set of quadratic residues modulo N). Thus, $r' = r$. Next, proof EQLOG-2(r, n, N') in step 4 of the commit phase (in conjunction with INTVL($r \in [0, N-1]$)) shows that, necessarily, the same value r is used to blind the signature on message m.

For the correct blinding, since it was proven that $\log_X m = \log_{g'} b\,(= r)$ for $X = \tilde{s}^e$, the receiver knows that he will be able to compute s as $s = \tilde{s}^r \bmod n$, once he has computed r. This holds in spite of the fact that r may be larger than the unknown quantity $\varphi(n)$, as this merely results in a "wrap-around."

The receiver will be able to retrieve r, either directly from the signer, or in time T by running the time-line's forced open algorithm. ∎

Third-party release. We see that the committer has to compute $s^{1/r} \bmod n$ in the blinding step (step 1). This corresponds to raising the signature s to the value $1/r \bmod \varphi(n)$. This is not a problem if the committer is also the signer, as he would know $\varphi(n)$. However, the above technique does not allow one party to commit to a signature that was generated by another party. In the full version of the paper, we show how to do this efficiently for RSA signatures. We now show how this can be done for Schnorr and DSA signatures, allowing the timed release of signatures by parties other than the signer. This may be a beneficial property in payment schemes, among other applications.

4.2 Timed Schnorr signatures

In order not to diverge too much from common notation, but still not to overload parameter names, we will be using an horizontal line above signature-specific names. Let $\overline{g}$ be a generator of a group of size $\overline{q}$, and let all computation, where applicable, be modulo $\overline{p}$, where $\overline{q}$ divides $\overline{p}-1$. We let $\overline{x} \in \mathbb{Z}_{\overline{q}}$ be the secret key, and $\overline{y} = \overline{g}^{\overline{x}}$ be the public key. Recall that a Schnorr signature is generated as follows. The signer selects $\overline{k} \in_R \mathbb{Z}_{\overline{q}}$, and computes $\overline{\rho} = \overline{g}^{\overline{k}} \bmod \overline{p}$ and $\overline{s} = \overline{k} + \overline{x}h(m, \overline{\rho}) \bmod \overline{q}$, where m is the message to be signed.

Commit phase. We assume for simplicity that the signer and the verifier have already done a time-line commitment TL(r), and that the committer knows a signature $(\overline{\rho}, \overline{s})$ on a message m known by both the committer and the receiver. (Note that the comitter does not need to be the signer in the following.) The committer and receiver perform the following steps:

1. *Application of blinding factor.* Let r be the blinding value time-committed to. Let $\tilde{s} = \overline{s}/r \bmod \overline{q}$ be the blinded signature, and $\tilde{g} = \overline{g}^r \bmod \overline{p}$ the blinded generator. The committer outputs $(\tilde{g}, \overline{\rho}, \tilde{s})$.

2. *Auxiliary commitments and proof of uniqueness.* This is performed identically to how it was performed for timed RSA release.

3. *Proof of correct blinding.* The verifier checks that $\tilde{g}^{\tilde{s}} = \overline{\rho}\overline{y}^{h(m,\overline{\rho})} \bmod \overline{p}$. The committer proves that $log_{\overline{g}}\tilde{g} = log_{g'}b\,(= r)$ using EQLOG-2$(r, \overline{p}, N')$, and that $r = log_{\overline{g}}\tilde{g}$ is in the right interval using INTVL$(r \in [0, N-1])$.

Open and forced open phases. The verifier obtains r from the signer or from the time-line Then, the verifier unblinds the signature: $\overline{s} = \tilde{s}r \bmod \overline{q}$, and outputs $(m, \overline{\rho}, \overline{s})$.

Theorem 3. *Under the hardness of the discrete logarithm problem and the generalized BBS assumption, the scheme above is a (T, t, ϵ) timed Schnorr signature scheme.*

The proof of this theorem is similar to that for RSA signatures (Theorem 2), and will be given in the full version of the paper.

4.3 Timed DSA signatures

We use similar notation here as above for Schnorr signatures. Let m be the (hashed) message to be signed. A DSA signature is generated by the signer selecting $\overline{k} \in_R \mathbb{Z}_{\overline{q}}$, computing $\overline{\rho} = \overline{g}^{\overline{k}} \bmod \overline{p}$, $\overline{\lambda} = \overline{\rho} \bmod \overline{q}$, and $\overline{s} = \overline{k}^{-1}(m + \overline{x}\overline{\lambda}) \bmod \overline{q}$. As for Schnorr signatures, the committer may be a different entity than the signer; in that case, however, we assume that he knows the auxiliary information $\overline{\rho}$.

Commit phase. We assume for simplicity that the signer and the verifier have already done a time-line commitment TL(r), and that the committer knows the unreduced signature $(\overline{\rho}, \overline{s})$ on a message m. The committer and receiver perform the following steps:

1. *Application of blinding factor.* Let r be the blinding value committed to. Let $\tilde{\rho} = \overline{\rho}^r \bmod \overline{p}$, and $\tilde{s} = \overline{s}/r \bmod \overline{q}$. The committer outputs $(\tilde{\rho}, \overline{\rho}, \overline{\lambda}, \tilde{s})$.

2. *Auxiliary commitments and proof of uniqueness.* This is performed identically to how it was performed for timed RSA and Schnorr release.

3. *Proof of correct blinding.* The verifier checks that $\tilde{\rho}^{\tilde{s}} \equiv_{\overline{p}} \overline{g}^m \overline{y}^{\overline{\lambda}}$. The committer proves that $log_{\overline{\rho}}\tilde{\rho} = log_{g'}b\,(= r)$ using EQLOG-2$(r, \overline{p}, N')$, and that $r = log_{\overline{\rho}}\tilde{\rho}$ is in the right interval using INTVL$(r \in [0, N-1])$.

Open and forced open phases. The value r is computed as above. Then, the verifier unblinds the signature: $\overline{s} = \tilde{s}r \bmod \overline{q}$. He outputs $(m, \overline{\lambda}, \overline{s})$.

Theorem 4. *Under the hardness of the discrete logarithm problem and the generalized BBS assumption, the scheme above is a (T, t, ϵ) timed DSA signature scheme.*

Proof in the full version of the paper.

Acknowledgements

Our constructions required a number of technical elements. We are indebted to Daniel Bleichenbacher, Don Coppersmith, Ari Juels and Carl Pomerance for pointing out inconsistencies and making insightful comments and suggestions along the way.

References

[Bao98] F. Bao. An efficient verifiable encryption scheme for encryption of discrete logarithms. In *Proc. CARDIS'98*, 1998.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[BCDvdG87] E. Brickell, D. Chaum, I. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret (extended abstract). In *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 156–166. Springer-Verlag, 1988, 16–20 August 1987.

[BG96] M. Bellare and S. Goldwasser. Encapsulated key escrow. In MIT/LCS/TR-688, 1996.

[BG97] M. Bellare and S. Goldwasser. Verifiable partial key escrow. In *Proc. ACM CCS*, pages 78–91, 1997.

[Ble00] D. Bleichenbacher. On the distribution of DSA session keys. Manuscript, 2000.

[BN00] D. Boneh and M. Naor. Timed commitments (extended abstract). In *Advances in Cryptology—CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[Bou00] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology—EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer-Verlag, 2000.

[BT99] F. Boudot and J. Traoré. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In *Proc. 2nd International Conference on Information and Communication Security*, volume 1726 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, 1999.

[Cha82] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Plenum Press, New York and London, 1983, 23–25 August 1982.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 21–25 August 1994.

[CEvdG87] D. Chaum, J. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Advances in Cryptology—EUROCRYPT 87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer-Verlag, 1988, 13–15 April 1987.

[CFT98] A. Chan, Y. Frankel, and Y. Thiounis. Easy come – easy go divisible cash. In *Advances in Cryptology—EUROCRYPT 98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575. Springer-Verlag, 1998.

[CM99] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes (extended abstract). In *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 414–430. Springer-Verlag, 1999.

[CP92] D. Chaum and T. Pedersen. Wallet databases with observers (extended abstract). In CRYPTO'92 [CRY92], pages 89–105.

[CRY92] *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993, 16–20 August 1992.

[DN92] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In CRYPTO'92 [CRY92], pages 139–147.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[GMP01] S. Galbraith, W. Mao, and K. Paterson. A cautionary note regarding cryptographic protocols based on composite integers. In HPL-2001-284, 2001.

[Mao98] W. Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Proc. Public Key Cryptography '98*, pages 27–42, 1998.

[May93] T. May. Timed-release crypto. In `http://www.hks.net.cpunks/cpunks-0/1460.html`, 1993.

[RSW96] R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. In MIT/LCS/TR-684, 1996.

[Sha84] A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1985, 19–22 August 1984.

[Sha95] A. Shamir. Partial key escrow: A new approach to software key escrow. In *Key Escrow Conference*, 1995.