# Timed Commitments

**(Extended Abstract)**

Dan Boneh[1] and Moni Naor[2]

[1] Stanford University, dabo@cs.stanford.edu
[2] Weizmann institute, naor@wisdom.weizmann.ac.il

**Abstract.** We introduce and construct *timed commitment* schemes, 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 without the help of the committer. An important application of our timed-commitment scheme is contract signing: two mutually suspicious parties wish to exchange signatures on a contract. We show a two-party protocol that allows them to exchange RSA or Rabin signatures. The protocol is *strongly fair*: if one party quits the protocol early, then the two parties must invest comparable amounts of time to retrieve the signatures. This statement holds even if one party has many more machines than the other. Other applications, including honesty preserving auctions and collective coin-flipping, are discussed.

## 1 Introduction

This paper introduces *timed commitments*. A timed commitment is a commitment scheme in which there is an optional forced opening phase enabling the receiver to recover (with effort) the committed value without the help of the committer. A regular commitment scheme consists of two phases: (i) The commit phase at the end of which the sender is bound to some value $b$, and (ii) The reveal phase, where the sender reveals $b$ to the receiver. Following the commit phase the sender should be bound to $b$, but the receiver should be unable to learn anything about $b$. A timed commitment has an additional forced opening phase, where the receiver computes a moderately hard function and recovers the value $b$ without the participation of the sender. As a result, the value $b$ remains hidden from the receiver for only a limited amount of time.

Timed commitments satisfy the following properties: (1) *verifiable recovery:* if the commit phase ends successfully, the receiver is convinced that forced opening will yield $b$. (2) *recovery with proof*: the receiver not only recovers $b$ but also a proof of its value, so that anyone who has the commitment (or the transcript of the commit phase) can verify that $b$ is the value committed *without going through a recovery process.* (3) *commitment immune against parallel attacks*: even if a receiver has many more processors than assumed, it cannot recover $b$ much faster than a single-processor receiver.

An important application of our timed-commitment scheme is contract signing: two mutually suspicious parties wish to exchange signatures on a contract.

We show a two-party protocol that allows them to exchange RSA, Rabin or Fiat-Shamir signatures (or any scheme where the signature is a power of a predetermined value mod a composite). Therefore, no special "infrastructure" is needed. One can use existing PKI and there is no need to modify the "semantics" of a signature.

Our contract signing protocol is based on gradual release of information (however, we do not release the signature "bit-by-bit"). Our protocol enjoys strong fairness: whatever one party may do (e.g. sending false information, stopping prematurely, etc.), the time it takes the other party to recover the signature is within a small constant of the time that it takes the misbehaving party to do so. Furthermore, our protocol is the first to resist parallel attacks. Even if one party has many more machines than the other, both parties need comparable amounts of time to recover the signature when one party aborts before the other.

Another important application of our scheme is honesty-preserving auctions. The auction participants who submit bids in sealed envelopes can have the sealing done using timed-commitments. This allows the auctioneer to convince all the participants that all the bids were considered, even if some of the parties wish to withdraw their bids.

Our timed-commitments can be used to obtain zero-knowledge in various settings, including the concurrent and the resettable settings. The most interesting application that requires all the properties of our commitment (provable recoverability and immunity to parallelization) is a three-round zero-knowledge protocol for NP in the timing model of [23].

We also confirm the folklore belief that two-party contract signing protocols require many rounds. We prove that if a protocol is to have bounded unfairness for the two parties (the ratio between the time it takes to recover a signature after early stopping) then it must take a number of rounds that is proportional to the security of the forging.

**Related Work** The problem of contract signing was the impetus of much of the early research on cryptographic protocols. Roughly speaking contract signing protocols can be partitioned into:

– Protocols that employ a trusted third-party as a referee or judge. The third party intervenes only if things go wrong. Examples include [2, 3, 7, 26, 33, 34]
– Protocols that are pure two-party (do not require a third party referee). Such protocols are based on the gradual release of signatures/secrets. Examples are [8, 24, 15, 18, 30].

Our method falls into the second category. The best scheme in this category is due to Damgard [18]. However, this scheme releases the actual signature bit-by-bit and hence is *not* immune to parallel exhaustive search attacks. If one party has access to more machines then the other then the protocol becomes unfair. This deficiency is common to all previously proposed schemes in this category.

Several recent papers focus on the trusted third party model with the goal of making it more fair, abuse free and accountable [26]. These goals are achieved by the strong fairness of our protocol.

**Timed primitives** Timed primitives previously came up in several contexts: Dwork and Naor [20] suggested moderately hard functions for "pricing via processing" in order to deter abuse of resources, such as spamming. Bellare and Goldwasser [4, 5] suggested "time capsules" for key escrowing in order to deter widespread wiretapping. A major issue there is to verify at escrow-time that the right key is escrowed. Similar issues arise in our work, where the receiver should make sure that at the end of the commit phase the value is recoverable.

Rivest, Shamir and Wagner [36] suggested "time-locks" for encrypting data so that it is released only in the future. This is the only scheme we are aware of that took into account the parallel power of the attacker. We employ a function similar to the one they suggested. However in their setting no measures are taken to verify that the puzzle can be unlocked in the desired time.

## 2 Timed commitments and timed signatures

We begin by defining our notions of *timed commitments* and *timed signatures.* We give efficient constructions for these primitives in the next section.

A $(T, t, \epsilon)$ *timed commitment* scheme for a string $S \in \{0,1\}^n$ enables Alice to give Bob a commitment to the string $S$. At a later time Alice can prove to Bob that the committed string is $S$. However, if Alice refuses to reveal $S$, Bob can spend time $T$ to forcibly retrieve $S$. Alice is assured that within time $t$ on a parallel machine with polynomially many processors, where $t < T$, Bob will succeed in obtaining $S$ with probability at most $\epsilon$. Formally, a $(T, t, \epsilon)$ timed commitment scheme consists of three phases:

**Commit phase:** To commit to a string $S \in \{0,1\}^n$ Alice and Bob execute a protocol whose outcome is a *commitment string* $C$ which is given to Bob.

**Open phase:** At a later time Alice may reveal the string $S$ to Bob. They execute a protocol so that at the end of the protocol Bob has a proof that $S$ is the committed value.

**Forced open phase:** Suppose Alice refuses to execute the open phase and does not reveal $S$. Then there exists an algorithm, called forced-open, that takes the commitment string $C$ as input and outputs $S$ and a proof that $S$ is the committed value. The algorithm's running time is at most $T$.

The commitment scheme must satisfy a number of security constraints:

BINDING: during the open phase Alice cannot convince Bob that $C$ is a commitment to $S' \neq S$.

SOUNDNESS: At the end of the commit phase Bob is convinced that, given $C$, the forced open algorithm will produce the committed string $S$ in time $T$.

PRIVACY: every PRAM algorithm $\mathcal{A}$ whose running time is at most $t$ for $t < T$ on polynomially many processors, will succeed in distinguishing $S$ from a random string, given the transcript of the commit protocol as input, with advantage at most $\epsilon$. In other words,

$$\left| \Pr[\mathcal{A}(\text{transcript}, S) = \text{"yes"}] - \Pr[\mathcal{A}(\text{transcript}, R) = \text{"yes"}] \right| < \epsilon$$

where the probability is over the random choice of $S$ and $R$ and the random bits used to create $C$ from $S$ during the commit phase.

Note that the privacy constraint measures the adversary's run time using a parallel computing model (a PRAM). Consequently, the privacy requirement ensures that even an adversary equipped with a highly parallel machine must spend at least time $t$ to forcibly open the commitment (with high probability). In other words, even an adversary with thousands of machines at his disposal cannot extract $S$ from $C$ in less than time $t$.

We define *timed signatures* analogously to timed commitments. A $(T, t, \epsilon)$ timed signature scheme enables Alice to give Bob a signature $S$ on a message $M$ in two steps. In the first step Alice commits to the signature $S$ and Bob accepts the commitment. At a later time Alice can completely reveal the signature $S$ to Bob. However, if Alice does not reveal the signature, Bob can spend time $T$ to forcibly retrieve the signature from the commitment. As before, Alice is assured that after time $t$, where $t < T$, Bob will not be able to retrieve the signature with probability more than $\epsilon$.

As before, a timed signature consists of three phases commit, open, and forced-open. The commit phase is identical to the commit phase of timed commitments. It results in Bob accepting a commitment string $C$. At a later time Alice may execute the open phase. At the end of the open phase Bob obtains a standard (message,signature) pair satisfying all the requirements of a digital signature. If the open phase is never executed Bob can run an algorithm whose run time is at most $T$ to forcibly extract the signature $S$ from the commitment $C$. In addition to soundness, a $(T, t, \epsilon)$ timed signature scheme must satisfy the following privacy requirement: all PRAM algorithms whose running time is at most $t$ will succeed in extracting $S$ from the commitment $C$ with probability at most $\epsilon$.

## 3 A timed commitment and timed signature scheme

As one can imagine, there are two main difficulties in building a timed commitment scheme. First, during the commit phase, the committer (Alice) must convince the verifier (Bob) that the forced open algorithm will successfully retrieve the committed value. This must be done without actually running the forced open algorithm. Second, we must ensure that even an adversary with thousands of machines cannot forcibly open the commitment much faster than a legitimate party with only one machine. To solve the later issue we base our scheme on a problem that appears to be inherently sequential: modular exponentiation. We use the fact that the best known algorithm for computing $g^{(2^m)} \bmod N$ takes $m$ *sequential* squarings. The surprising fact is that there is an efficient zero-knowledge protocol, with a running time of $O(\log m)$, enabling the committer (Alice) to prove to the verifier (Bob) that the result of these $m$ squarings will produce the committed message $M$. This proof is done during the commit phase and is at the heart of our timed commitment scheme.

Let $T = 2^k$ be an integer. We build a timed commitment scheme where it takes $T = 2^k$ modular multiplications to forcibly retrieve the committed string. The commit phase takes $O(k)$ modular exponentiations. We envision $k$ as typically being in the range $[30, \ldots, 50]$. This way the forced open algorithm can take a few hours, or a few days depending on the requirements.

**Setup:** Let $n$ be a positive integer representing a certain security parameter. The committer generates two random $n$-bit primes $p_1$ and $p_2$ such that $p_1 = p_2 = 3 \bmod 4$. He computes $N = p_1 p_2$. The committer publishes $\langle N \rangle$ as a public key (alternatively he could send $N$ along with every commitment). He keeps the factors $\langle p_1, p_2 \rangle$ secret. The same modulus is used for all commitments.

**Commit phase:** The committer wishes to commit to a message $M$ of length $\ell$. The committer (Alice) and the verifier (Bob) perform the following steps:

*Step 1:* The committer picks a random $h \in \mathbb{Z}_N$. Next, the committer computes $g = h^{(\prod_{i=1}^{r} q_i^n)} \bmod N$ where $q_1, q_2, \ldots, q_r$ is the set of all primes less than some bound $B$. For example, one could take $B = 128$. When the verifier receives $h$ and $g$ he verifies that $g$ is constructed properly from $h$. At this point the verifier is assured that the order of $g$ in $\mathbb{Z}_N^*$ is not divisible by any primes less than $B$.

*Step 2:* The committer computes the value $u = g^{2^{2^k}} \bmod N$. She computes $u$ by first computing $a = 2^{2^k} \bmod \varphi(N)$ and then computing $u = g^a \bmod N$.

*Step 3:* Next, the committer hides the message $M$ using a pseudo random sequence generated by the BBS generator [9] whose tail is $u$. In other words, the committer hides the bits of $M$ by Xoring them with the LSB's of successive square roots of $u$ modulo $N$. More precisely, for $i = 1, \ldots, \ell$ we set $S_i = M_i \oplus \mathrm{lsb}(g^{2^{(2^k - i)}} \bmod N)$. Let $S = S_1 \ldots S_\ell \in \{0,1\}^\ell$. The commitment string is defined as $C = \langle h, g, u, S \rangle$. The committer sends $C$ to the verifier.

*Step 4:* The committer must still convince the verifier that $u$ is constructed properly, i.e. $u = g^{2^{2^k}} \bmod N$. To do so the committer constructs the following vector $W$ of length $k$:

$$W = \left\langle g^2, g^4, g^{16}, g^{256}, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^k}} \right\rangle \pmod N$$

She sends $W$ to the verifier. Let $W = \langle b_0, \ldots, b_k \rangle$. For each $i = 1, \ldots, k$ the committer proves in zero-knowledge to the verifier that the triple $(g, b_{i-1}, b_i)$ is a triple of the form $(g, g^x, g^{x^2})$ for some $x$. This convinces the verifier that $W$ is constructed properly. By verifying that the last element in $W$ is equal to $u$ the verifier is assured that indeed $u = g^{2^{2^k}} \bmod N$.

Each of these $k$ proofs take four rounds and they can all be done in parallel. These proofs are based on a classic zero-knowledge proof that a tuple $\langle g, A, B, C \rangle$ is a Diffie-Hellman tuple [13]. Let $q$ be the order of $g$ in $\mathbb{Z}_N^*$, and let $R$ be a security parameter. The complete protocol for proving integrity of $W$ is as follows: (unless otherwise specified, all arithmetic is done modulo $N$)

**Step 1:** The verifier picks random $c_1, \ldots, c_k \in \{0, \ldots, R\}$ and uses a regular commitment scheme to commit these values to the committer. For security against an infinitely powerful committer the verifier could use a commitment

scheme that is information theoretically secure towards the committer.
**Step 2:** The committer picks random $\alpha_1, \ldots, \alpha_k \in \mathbb{Z}_q$ and computes $z_i = g^{\alpha_i}$ and $w_i = b_{i-1}^{\alpha_i}$ for $i = 1, \ldots, k$. She sends all pairs $\langle z_i, w_i \rangle_{i=1}^k$ to the verifier.
**Step 3:** The verifier opens the commitment in step 1 and reveals $c_1, \ldots, c_k$ to the committer.
**Step 4:** The committer responds with $y_i = c_i \cdot 2^{2^{i-1}} + \alpha_i \bmod q$ for all $i = 1, \ldots, k$.
**Step 5:** The verifier checks that for all $i = 1, \ldots, k$:

$$g^{y_i} \cdot b_{i-1}^{-c_i} = z_i \qquad \text{and} \qquad b_{i-1}^{y_i} \cdot b_i^{-c_i} = w_i$$

and rejects if any of these equalities does not hold.

The next two lemmas state the soundness and zero-knowledge properties of the above protocol.

**Lemma 1.** *Let $q$ be the order of $g$ in $\mathbb{Z}_N^*$ and let $d$ be the smallest prime divisor of $q$. If $W$ is constructed incorrectly then the committer will succeed in fooling the verifier with probability at most*[1] $k \cdot [\frac{1}{\min(d,R)} + o(\frac{1}{R})]$.

**Lemma 2.** *The above protocol is zero-knowledge. That is, there exists a simulator that produces a perfect simulation of the transcript for any verifier.*

The proofs of the two lemmas follow standard techniques and can be found in [13]. Recall that in Step 1 of the commitment protocol the verifier is convinced that the smallest prime divisor of $q$ (the order of $g$ in $\mathbb{Z}_N^*$) is larger than $B$. Hence, with each invocation of the protocol, the committer has a chance of at most $1/B$ in fooling the verifier. In this context, security of $2^{-70}$ is sufficient. Hence, taking $B = 128$, this level of security is obtained if the committer and verifier execute this protocol 10 times. These executions can be done in parallel.

Note that in Step 5, the verifier computes $4k$ exponentiations. However, using simultaneous multiple exponentiation [31, p. 618] the two exponentiations on the left hand side of each equality can be done for approximately the cost of one exponentiation. Hence, in reality, the verifier does work equivalent to $2k$ exponentiations. Recall that $k$ is typically in the range $[30, 50]$. Thus, counting 10 repetitions of the proof, the commit protocol requires at most a total of 1000 exponentiations on the verifier.

**Open phase:** Recall that the commitment string is $C = \langle h, g, u, S \rangle$. We know that $g = h^{(\prod_{i=1}^r q_i^n)} \bmod N$ where $q_1, q_2, \ldots, q_r$ is the set of all primes less than some bound $B$. In the open phase the committer (Alice) and verifier (Bob) execute the following protocol:

*Step 1:* Alice sends $v' = h^{2^{(2^k - \ell)}} \bmod N$ to the verifier (Bob). Bob computes $v = (v')^{\prod_{i=1}^r q_i^n} \bmod N$. This ensures that $v$ has odd order. Bob then verifies that $v^{2^\ell} = u \bmod N$. At this point Bob has a $2^\ell$'th root of $u$. Being a $2^\ell$'th root

[1] The exact error bound is $k \cdot \left[\frac{1}{d} + \frac{\beta(d-\beta)}{d \cdot R^2}\right]$ where $\beta = R \bmod d$, $0 \leq \beta \leq d$. This expression is the maximum probability that a malicious prover succeeds in guessing $c_i \bmod d$ where $c_i$ is random in $\{0, \ldots, R\}$.

of $u$ and having odd order ensures that $v$ is in the subgroup generated by $g$.
*Step 2:* Bob constructs the $\ell$-bit BBS pseudo-random sequence $R \in \{0,1\}^\ell$ starting at $v$. That is, for $i = 1, \ldots, \ell$ Bob sets $R_i$ to be the least significant bit of $v^{2^{(\ell-i)}}$. The message $M$ is then $M = R \oplus S$.

With an honest committer the open protocol clearly produces the committed message $M$. We show that the commitment is binding by showing that $M$ is the only possible outcome of the open protocol.

**Lemma 3.** *The commitment is binding. In other words, given a commitment $\langle h, g, u, S\rangle$ the committer can open the commitment in one way only.*

*Proof.* Due to the test in Step 1 Bob obtains from Alice $v'$ which leads to $v \in \mathbb{Z}_N$ satisfying $v^{2^\ell} = u \bmod N$. Furthermore, $v$ has odd order in $\mathbb{Z}_N^*$. Recall that during the commit phase the verifier is assured that $g$ has odd order in $\mathbb{Z}_N^*$ and that $u$ is in the subgroup generate by $g$. Since the subgroup has odd order, $u$ has a unique $2^\ell$'th root *in the subgroup*. Denote this unique $2^\ell$'th root by $v_0$. Then $v_0 = g^{2^{(2^k-\ell)}} \bmod N$. Now, observe that in $\mathbb{Z}_N^*$ there can be at most one $2^\ell$'th root of $u$ of odd order. Since both $v$ and $v_0$ are such roots we must have $v = v_0$. Consequently, there is a unique $v' \in \mathbb{Z}_N^*$ that will pass the test in Step 1. Hence, Alice is bound to a unique message $M$. $\square$

**Forced open phase:** In case the committer never executes the open phase, the verifier can retrieve the committed value $M$ himself by computing $v$ as $v = g^{2^{(2^k-\ell)}} \bmod N$ using $(2^k - \ell)$ squarings mod $N$.

We now prove that the above scheme satisfies the security properties of a timed commitment scheme. The only property that remains to be proved is *privacy*: no PRAM algorithm can obtain information about the committed string in time significantly less than the time it takes to compute $2^k$ squarings. The proof of security relies on the following complexity assumption:

$(n, n', \delta, \epsilon)$ **generalized BBS assumption:**

For $g \in \mathbb{Z}$ and a positive integer $k > n'$ let $W_{g,k} = \langle g^2, g^4, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^k}}\rangle$. Then for any integer $n' < k < n$ and any PRAM algorithm $\mathcal{A}$ whose running time is less than $\delta \cdot 2^k$ we have that

$$\Big| \Pr\left[\mathcal{A}(N,\, g,\, k,\ W_{g,k} \bmod N,\, g^{2^{2^{k+1}}}) = \text{“yes”}\right] - \\ \Pr\left[\mathcal{A}(N,\, g,\, k,\, W_{g,k} \bmod N,\, R^2) = \text{“yes”}\right] \Big| < \epsilon$$

where the probability is taken over the random choice of an $n$-bit RSA modulus $N = p_1 p_2$ where $p_1, p_2$ are equal size primes satisfying $p_1 = p_2 = 3 \bmod 4$, an element $g \in \mathbb{Z}_N$, and $R \in \mathbb{Z}_N$.

The assumption states that given $W_{g,k}$, the element $g^{2^{2^{k+1}}} \bmod N$ is indistinguishable from a random quadratic residue for any PRAM algorithm whose running time is much less than $2^k$. The parallel complexity of exponentiation modulo a composite has previously been studied by Adleman and Kompella [1] and Sorenson [39]. These results either require a super polynomials number of processors (larger than the time to factor), or give small speed ups that do not affect the generalized BBS assumption. We note that the sequential nature of exponentiation modulo a composite was previously used for time-lock cryptography [36] and benchmarking [10].

The generalized BBS assumption is sufficient for proving privacy of the scheme. This is stated in the next lemma whose proof is omitted due to lack of space.

**Theorem 1.** *Suppose the $(n, n', \delta, \epsilon)$ generalized BBS assumption holds for certain $\delta, \epsilon > 0$. Then, for $k > n'$, the above scheme is a $(T, t, \epsilon)$ timed commitment scheme with $t = \delta \cdot 2^k$ and $T = M(n) \cdot 2^k$ where $M(n)$ is the time it takes to square modulo an $n$-bit number.*

**Efficiency improvements** The timed commitment scheme described above can be made more efficient as follows:

- Rather than use a random element $g \in \mathbb{Z}_N$ whose order is close to $N$ we can use a $g$ of much smaller order. To do so, we choose $N = p_1 p_2$ where $q_1$ divides $p_1$ and $q_2$ divides $p_2$ where $q_1, q_2$ are distinct $m$ bit primes, and $m \ll \log_2 N$. We then use an element $g$ in $\mathbb{Z}_N^*$ of order $q = q_1 q_2$. Since $g$ has small order the exponentiations in Step 4 of the commitment protocol take far less time.
- Suppose the committer needs to repeatedly commit a message to the same verifier Bob. In this case the protocol can be improved significantly. During the Setup phase, the committer picks a modulus $N = p_1 p_2$ where $p_1$ and $p_2$ are strong primes, i.e. $\frac{p_1-1}{2}, \frac{p_2-1}{2}$ are prime. The committer and the verifier then execute a protocol due to Camenisch and Michels [11] to convince the verifier in zero-knowledge that $N$ is the product of two strong primes and that the smallest prime factor of $N$ is at least $m$ bits long for some predetermined $m$ (e.g. $m = 70$). This protocol is only executed once in order to validate the public key $N$. Step 1 of the commitment protocol is now replaced by the committer picking a random $h \in \mathbb{Z}_N^*$ and setting $g = h^2 \bmod N$. Let $q$ be the order of $g$ in $\mathbb{Z}_N^*$. Since $N$ is the product of strong primes greater than $2^m$ the verifier is assured that the smallest prime factor of $q$ is greater than $2^{m-1}$. Hence, by Lemma 1 the protocol for verifying integrity of $W$ in Step 4 of the commitment protocol need only be run once (rather than multiple times as discussed above).

### 3.1 A timed signature scheme

A timed signature scheme can be easily built out of our timed commitment scheme and a regular signature scheme. Let $(\sigma, V, G)$ be a signature scheme ($\sigma$ takes a message and a private key and generates a signature, $V$ takes a signature and a public key and verifies the signature, and $G$ is a public/private key pair generator). The $(T, t, \epsilon)$ timed signature scheme is as follows:

**Setup:** The signer generates a public/private key pair $\langle Pub, Pr \rangle$ using algorithm $G$. The signer's public key is $\langle Pub \rangle$. He keeps $\langle Pr \rangle$ secret.

**A valid signature:** A valid signature on a message $M$ is a tuple $SIG = \langle S, C, Sig \rangle$ where (1) $C$ is a commitment string generated by the timed commitment scheme when committing to the string $S$, and (2) $Sig$ verifies using the public key $Pub$ as a valid signature on $\langle M, C \rangle$.
**Commit phase:** The signer picks a random secret string $S$ and uses the timed commitment scheme to commit $S$ to the verifier. Let $C$ be the resulting commitment string given to the verifier. The signer uses her private key $Pr$ to sign the message $\langle M, C \rangle$. Let $Sig$ be the resulting signature. The full commitment string given to the verifier is $\langle C, Sig \rangle$.
**Open phase:** The signer reveals $S$. The verifier obtains a complete valid signature as $\langle S, C, Sig \rangle$.
**Forced open:** Use the forced open algorithm provided by the timed commitment to retrieve $S$.

This signature scheme has the feature (or bug) that once the commit phase is done, the verifier can convince a third party that the signer is about to give him a signature on $M$. Indeed, the value $Sig$ in the commitment string given to the verifier could have only come from the signer. This is called an *abuse* of the protocol [26]. In Section 4.2 we construct a timed signature scheme so that after the commit phase is done the verifier cannot convince a third party that he has been talking to the signer. This is a desirable property when using timed signatures for contract signing.

## 4 Contract signing

In this section we show how to use our timed primitives for contract signing and fair exchange of signatures. We show how to enable two untrusting parties, Alice and Bob, to exchange a signature on a joint contract. Neither party is willing to sign the contract before the other. Any timed-signature scheme can be used to solve the problem without relying on any infrastructure beyond a standard CA. The main difference between previous gradual-disclosure solutions and ours is that by using timed signatures Alice need not worry that Bob has ten times more machines than her. Timed signatures are resistant to parallel attacks, whereas previous proposals for gradual release of secrets become unfair as soon as one party has more machines than the other.

We begin by specifying the desired properties of a contract signing protocol. We then show that the timed commitment scheme of the previous section gives an especially efficient solution.

### 4.1 Contract Signing Definitions

A contract signing protocol allows two parties, $A_0$ and $A_1$, to exchange signatures on a contract $C$. Assume that $A_0$ and $A_1$ have established public keys $P_0$ and $P_1$ respectively. For a given contract $C$ the two parties exchange messages. At the end of the protocol each party $A_b$ (for $b \in \{0, 1\}$) has a signature $S_{1-b}(C)$ such that a third party (contract verifier) that is given $S_{1-b}(C)$ as well as $P_0$

and $P_1$ can verify the signature. More precisely, for a contract signing protocol to be reasonable at all we need the following two conditions, which are standard in signature schemes:

**Completeness** With overwhelming probability the *signature verification algorithm* outputs "accept" on a signature that is the result of a correct (by both parties) execution of the protocol.

**Unforgeability** For a properly generated $P_b$ and for any contract $C$, unless $A_b$ participated in the contract signing protocol or $C$, the probability that any probabilistic polynomial-time adversary $\mathcal{A}$ succeeds in finding a signature $S$ and key $P_{1-b}$ such that verifier accepts $(C, S, P_b, P_{1-b})$ is negligible.

In addition to the normal operations (when no cheating by the other party occurs) the protocol designer should also provide to each party $A_b$ a **forced signature opening** algorithm $R_b$: given the private information of $A_b$ plus the messages exchanged with $A_{1-b}$ algorithm $R_b$ tries to produce $S_{1-b}(C)$. The time that $R_b$ is allowed to perform the recovery may be given as a parameter.

In order to define fairness we view a contract signing protocol as a game, where the goal of each party is to obtain a signature on a contract that is considered valid by the *signature verification algorithm* (for simplicity and wlog we assume that the verification algorithm is deterministic), without revealing its own signature.

**Definition 1.** *We say that a protocol is* $(c, \epsilon)$-fair *if the following holds: for any time $t$ smaller than some security parameter and any adversary $\mathcal{A}$ working in time $t$ as party $A_b$: let $\mathcal{A}$ choose a contract $C$ and run the contract signing protocol with party $A_{1-b}$. At some point $\mathcal{A}$ aborts the protocol and attempts to recover a valid signature $S_{1-b}(C)$. Denote $\mathcal{A}$'s probability of success by $q_1$. Suppose now that party $A_{1-b}$ runs the forced signature opening $R_{1-b}$ for time $c \cdot t$ and let $q_2$ be the probability of recovering $S_b(C)$. Then $q_1 - q_2 \leq \epsilon$.*

The protocols in this section are $(2, \epsilon)$ fair for a negligible $\epsilon$. That is, suppose $A_b$ aborts the protocol at some point and then recovers $S_{1-b}(C)$ in time $t$. Then $A_{1-b}$ can recover the signature $S_b(C)$ in time $2t$. Smaller values of $c$ can be achieved as discussed in Section 4.3.

**Strong Fairness:** One problem with the above definition is that it leaves open the possibility that one party $A_b$ would be able to convince a third party that $A_{1-b}$ is in the process of signing the contract $C$ (even without the given signature verification algorithm), whereas $A_b$ does not have the signature $S_b(C)$. This type of unfairness is referred to as "abusing" [26, 38]. We propose a stronger requirement of fairness: if the adversary $\mathcal{A}$ invests time much smaller than the threshold $T$ allowing both parties to recover the desired signatures, then the transcript of the conversation is useless: there is simulator (operating in time proportional to the running-time of $\mathcal{A}$) that can output a transcript that is indistinguishable to any machine operating in time less than the threshold $T$. In other words, the protocol is zero-knowledge to an adversary not willing to invest the recovery time (in which case the full signature is extractable.)

### 4.2 The strongly fair contract signing protocol

We now describe a signature exchange protocol that enables two parties, Alice and Bob, to exchange Rabin signatures on a contract $C$. The protocol works as follows:

**Setup:** Alice generates an $N_a$ as in the timed commitment scheme of Section 3. She will use $\langle N_a \rangle$ for all her contract signings. Similarly Bob generates $\langle N_b \rangle$. They agree on $k$ as a security parameter, e.g. $k = 40$.

**Valid signature:** Alice's signature on $C$ is a regular Rabin signature modulo $N_a$. That is, let $H \in \mathbb{Z}_{N_a}$ be the hash of $C$ properly padded prior to signing (e.g. according to PKCS1 or Bellare-Rogaway [6]). Alice's signature on $C$ is $S = H^{1/2} \bmod N_a$. Bob's signature on $C$ is defined analogously modulo $N_b$.

**Init:** To sign a contract $C$ the protocol begins with Alice picking a random $g_a \in \mathbb{Z}_N$ and generating a vector

$$W_{alice} = \left\langle g_a^2, g_a^4, g_a^{16}, g_a^{256}, \ldots, g_a^{2^{2^i}}, \ldots, g_a^{2^{2^k}} \right\rangle \pmod{N_a}$$

Alice can constructs this vector efficiently by first reducing all the exponents modulo the order of $g_a$. She sends $W_{alice}$ to Bob. Next, she convinces Bob that $W_{alice}$ is constructed correctly, i.e. $W_{alice} = \langle u_0, \ldots, u_k \rangle$ where $u_i = g_a^{2^{2^i}}$. She does so using the zero-knowledge protocol described in the commit phase in Section 3.

Let $\langle v_0, \ldots, v_k \rangle$ be the square roots modulo $N_a$ of the elements in $W_{alice}$. Alice computes the Rabin signature on $C$, namely Alice computes $S = H^{1/2} \bmod N_a$. She sends $V = S \cdot (v_0 \cdots v_k) \bmod N_a$ to Bob. Bob verifies validity of $V$ by checking that $V^2 = H \cdot (u_0 \cdots u_k) \bmod N_a$.

Bob initializes his contribution to the protocol by doing the analogous operations modulo $N_b$.

**Iteration:** from now on Alice and Bob take turns in revealing the square roots of elements in $W_{alice}$ and $W_{bob}$. Alice begins by revealing $v_k^{(alice)}$, the square root modulo $N_a$ of the last element in $W_{alice}$ (namely $u_k^{(alice)}$). Bob responds by revealing $v_k^{(bob)}$, the square root modulo $N_b$ of the last element in $W_{bob}$ (namely $u_k^{(bob)}$). Next, Alice reveals the square root of $u_{k-1}^{(alice)}$ and Bob responds by revealing the square root of $u_{k-1}^{(bob)}$. This continues for $2k$ rounds until the square roots of all elements in $W_{alice}$ and $W_{Bob}$ are revealed. At this point Bob can easily obtain Alice's signature on $C$ by computing $V/(v_0 \cdots v_k) \bmod N_a$, where $V, v_0, \ldots, v_k$ are the values sent from Alice. Alice obtains Bob's signature on $C$ by doing the same on the values sent from Bob.

**Forced Signature Opening:** suppose Bob aborts the protocol after Alice reveals only $m < k$ square roots. Then Bob has $v_k, \ldots, v_{k-m+1} \in \mathbb{Z}_{N_a}$ that are the square roots of $u_k^{(alice)}, \ldots, u_{k-m+1}^{(alice)}$. He can compute the remaining square roots by computing $v_i = g^{2^{2^i-1}} \bmod N_a$ for all $i = 0, \ldots, m$. This requires approximately $2^m$ modular multiplications. He then obtains Alice's signature on $C$. Based on the generalized BBS assumption one can show that Bob cannot produce the signature any faster: this will imply that he can distinguish between $u_{k-m}$ and a random quadratic residue in $\mathbb{Z}_{N_a}$. Fortunately, Alice can also obtain Bob's signature on $C$ in roughly the same amount of time. Alice has to compute the square root of one more element than Bob did. Namely, she has to compute the square roots of $u_0^{(bob)}, \ldots, u_{m+1}^{(bob)}$. She does so by computing $v_i = g^{2^{2^i-1}} \bmod N_b$ for all $i = 0, \ldots, m+1$. This requires approximately $2^{m+1}$ modular multiplications. Thus her work load is roughly twice that of Bob's. Consequently, Bob does not gain much from prematurely aborting the protocol.

The following lemma shows that the above protocol is fair. The proof is along the lines of the above discussion, and is omitted due to space limitations.

**Lemma 4.** *Suppose the generalized BBS assumption holds for some parameters $(n, n', \delta, \epsilon)$. Say Bob aborts the protocol after only $n' < m < 2k$ rounds. He then recovers the complete signature in time $T$. Then Alice can obtain the complete signature in at most expected time $2T \cdot \epsilon/\delta$. The same holds if Alice aborts the protocol first.*

Note that even if Bob has many more machines than Alice he cannot gain much since, by assumption, parallelism does not speed up modular exponentiation. As long as Bob's machines run at approximately the same speed as Alice's machines, fairness is preserved.

The protocol also satisfies the strong fairness properties defined in the previous section. By the generalized BBS assumption, the value $u_k^{(alice)}$ looks random to a third party whose whose running time is much less than $2^k$. Since the proof of validity of $W_{alice}$ is zero-knowledge, Bob cannot use it to convince a third party that $W_{alice}$ is well formed. As a result, Bob can easily simulate the value $V$ by picking $V \in \mathbb{Z}_{N_a}$ at random, and setting $u_k = V^2/(H \cdot u_0 \cdots u_{k-1})$. Hence, Bob can simulate himself all the information he got from Alice during the commit phase. Consequently, suppose Bob misbehaves during the protocol (given the zero-knowledge proofs of consistency the only real bad behavior on Bob's part is early stopping), but Bob is able to convince with non-negligible probability a third party verifier that he is executing a contract signing protocol with Alice. Then Alice can produce a signature on the contract in time proportional to the running time of Bob and the third party verifier.

*Remark 1.* Although the above description requires $2k$ rounds, it is easy to cut it down to $k$ rounds. Simply make each party send two square roots per turn (rather than one). The advantage of early stopping is unchanged and rotates from party to party.
We also point out that any square root based signature can be used, e.g. the Fiat-Shamir method [25], or a non-oracle based one such as a variant of the Dwork-Naor scheme [21]. In the next section we show how to incorporate RSA signatures into the fair exchange scheme.

### 4.3 Extensions

**RSA Signatures** The scheme in Section 4.2 enables two parties to exchange Rabin signatures. We describe a simple extension enabling fair exchange of RSA signatures. Let $(N_a, e)$ be the public key of Alice. The difference with respect to Rabin signatures is that now

$$W_{alice} = \left\langle g_a^{e\cdot 2}, g_a^{e\cdot 4}, g_a^{e\cdot 16}, g_a^{e\cdot 256}, \ldots, g_a^{e\cdot 2^{2^i}}, \ldots, g_a^{e\cdot 2^{2^k}} \right\rangle \pmod{N_a}$$

Note that the generalized BBS assumption implies that the next element in this sequence is indistinguishable from random in time $\delta 2^k$, since it is easy to transform a sequence

$$\left\langle g_a^2, g_a^4, g_a^{16}, g_a^{256}, \ldots, g_a^{2^{2^i}}, \ldots, g_a^{2^{2^k}} \right\rangle \pmod{N_a}$$

into $W_{alice}$ by $k$ parallel exponentiations in $e$. Showing that the vector $W_{alice}$ is constructed correctly is similar to what was done previously: for any $1 \leq i \leq k$ Alice should prove that the triple $(g, b_{i-1}, b_i)$ is of the form $(g, g^{ex}, g^{ex^2})$ for a given $e$. This can be done by the same protocol that proves triples of the form $(g, g^x, g^{x^2})$. Simply run the protocol on the triple $(g, g^{ex}, g^{e^2x^2})$ and then verify that $b_i$ is the $e$th root of $g^{e^2x^2}$.

The contract signing protocol proceeds along the same lines as before. For $0 \leq i \leq k$ let $u_i = g_a^{e\cdot 2^{2^i}}$, let $v_i = u_i^{1/e} \bmod N_a = g_a^{2^{2^i}}$, i.e. the $v_i$'s are defined as $e$th roots of the $u_i$'s (instead of square roots, as above). Let $H$ be the value to be signed by the RSA signature, i.e. the goal of the recipient it to obtain $H^{1/e} \bmod N_a$. The $v_i$'s mask $H^{1/e} \bmod N_a$ — Alice gives Bob $V = H^{1/e} \cdot v_0 \cdot v_1 \cdots v_k$. As before, the $v_i$'s are released one-by-one. The validity of each $v_i$ is easy to verify by comparing $v_i^e$ to $u_i$.

To argue the security of the scheme we assume the generalized BBS Assumption as well as the usual RSA one (that it is hard to extract $e$th roots). Based on these two assumptions, given $u_0, u_1, \ldots u_i$ it is hard to distinguish between $u_{i+1}$ and a random value and it is hard to compute $v_{i+1}$. Therefore in case of early stopping $i$ steps from the end it is impossible to find the RSA signature, $H^{1/e} \bmod N_a$, more efficiently than to compute $g^{2^{2^i}}$. Furthermore, a simulator can efficiently create an indistinguishable conversation.

**Other ratios** The signature exchange protocol of Section 4.2 achieves a fairness ratio of $c = 2$. That is, if Alice aborts the protocol, Bob has to do *twice* as must work as Alice to obtain the signature. The protocol easily generalizes to provide smaller fairness ratios as well, at the cost of increasing the number of rounds. For example, one can define $W_{alice}$ as

$$W_{alice} = \left\langle g_a^{2^{c_0}}, g_a^{2^{c_1}}, g_a^{2^{c_2}}, \ldots g_a^{2^{c_k}} \right\rangle \pmod{N_a}$$

where $c_0 = 1$ and $c_i = c_{i-1} + c_{i-2}$ for $i = 1, \ldots, k$. The proof of validity of $W_{alice}$ given in Section 3 must be changed accordingly. This approach gives a fairness ratio of $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$. Even smaller values can be obtained by other such recurrences. The downside is that to obtain an initial security of $2^k$ the protocol must take $\log_\alpha 2^k$ rounds, as opposed to only taking $k$ rounds as in Section 4.2.

## 5 More applications

We now describe several other applications of timed commitments. In all applications, we assume that the parties involved use clocks, but the adversary has control over the scheduling and the clocks. However, the adversary must satisfy the $(\alpha, \beta)$ constraint (for $\alpha < \beta$) of clocks [23]: for any two (possibly the same) non-faulty parties $P_1$ and $P_2$, if $P_1$ measures $\alpha$ elapsed time on its local clock and $P_2$ measures $\beta$ elapsed time on its local clock, and $P_2$ begins its measurement in real time after $P_1$ begins, then $P_2$ will finish after $P_1$ does. An $(\alpha, \beta)$ constraint is implied by many reasonable assumptions on the behavior of clocks in a system (*e.g.* the linear drift assumption). We assume that $\alpha$ is large enough that one party can send a message and expect a response in time $\alpha$, and $\beta$ is smaller than the security parameter of the commitment scheme.

**Collective coin-flipping** We have two parties $A$ and $B$ who want to flip a coin in such a matter that (i) the value of the coin is unbiased and well defined even if one of the parties does not follow the protocol (if both of them don't follow, then it is a lost case) (ii) if both parties follow the protocol, then they agree on the same value for the coin.

Consider the protocol where one party commits to a bit and the other guesses it and the result is the Xor of the two bits. The problem with this simple protocol is that one party knows the results before the other and can quit early (if it doesn't like the result), thus biasing the result. Indeed Cleve[14] has shown that for any $k$-round protocol one of the parties can bias the coin with at least $1/k$ (this bound was improved in [16] to $1/\sqrt{k}$).

What we show now is a protocol that works in the $(\alpha, \beta)$ timing model using our timed commitment and (i) the coin has only negligible bias. (ii) the number of rounds is constant.

1. $A$ to $B$: pick a random bit $b_A \in \{0, 1\}$ and time-commit to $b_A$.
2. $B$ to $A$: pick a random bit $b_B \in \{0, 1\}$ and send to A
3. $A$ to $B$: open $b_A$.

The collective coin is $b_A \oplus b_B$.
*Timing*: $A$ makes sure that $B$'s message arrives within time $\alpha$ from the beginning of Step 1.
*Forced opening*: if $A$ does not open $b_A$ at Step 3, then $B$ uses the forced opening procedure to extract $b_A$ and sets the coin to $b_A \oplus b_B$.

It is easy to verify that if $B$ can bias the coin, then it can guess the value of a timed-committed bit in time smaller than the security parameter of the scheme. On the other hand $A$ can try to influence the bit only by not opening $b_A$; however this is defeated by the forced opening.

**Honesty-Preserving Auctions** In a second-price (Vickrey) auction participants submit their bids to an item. The winner is the highest bidder but pays the second highest bid. A problem with running these auctions is establishing the trust in the auctioneer: how can the winner be assured that the auctioneer hasn't introduced a 2nd highest bid which is just $\epsilon$ less than the winning one?

Recent work focuses on many aspects of uncheatable auctions [32]. Here we are interested mostly in the *honesty-preserving* aspect of the protocol, i.e. making sure that the auctioneer is not changing some bids as a function of others. At the end of the protocol all the participants will know all the bids.
One simple solution is as follows:

1. The participants submit their bids by committing to their values. Here care must be taken to make the commitments *non-malleable* (see [19]).
2. When all of the commitments are in, the auctioneer posts them on a bulletin board. The participants should verify that their hidden bids were posted.
3. The participants then open their commitments. The auctioneer posts the results and everyone can verify that the auction was conducted properly

However, there is a problem with this solution: what if one (or more) participant refuses to open their commitment? If they are simply ignored, an auctioneer can plant several bogus bids with different values and open only those lower then the winning bid.

The timed commitment of Section 3 solves the problem. In Step 1 participants commit to their bid using a timed commitment. The bidders verify that that within time $\alpha$ the bid is posted, i.e. that from Step 1 to Step 2 no more than $\alpha$ time elapses. If in Step 3 a participant does not open its commitment, the auctioneer can "force open" the commitment using the forced opening algorithm.

Note that it is important for the commitment to have the soundness property (i.e. that at the end of the commit phase it is clear that forced open would work), otherwise the other bidders would not be convinced that it was properly opened.

**Zero-Knowledge** We now briefly discuss the application of our timed commitments for achieving zero-knowledge in various settings. Consider *concurrent zero-knowledge*: Several parties who are *not* mutually aware of each other and may lack coordination are simultaneously engaged in zero-knowledge protocols. This settings received much attention recently (see [23, 29, 35, 17].)The problem is in showing that the composed protocols are zero-knowledge in total. If the adversary controls the scheduling then it can create nested interactions that make the simulator's life difficult[2]. However, as suggested in [23], if the adversary is $(\alpha, \beta)$-restricted, then it is possible to obtain a constant round zero-knowledge protocol. Our timed commitments can be used for the verifier to commit to its queries. The verifier should accept only if the prover send its own commitments within time $\alpha$. The simulator force opens them and knows what to send as the prover.

Very recently Dwork and Naor [22] constructed *Zaps* - two-round witness indistinguishable proof systems - for any language in NP. Using timed-commitments with properties such as ours (verifiable recovery) they were able to show a three-round (concurrent) zero-knowledge proof system for all langauges in NP (in the $(\alpha, \beta)$-timing model.) The zap was used to prove that the commitment is proper

[2] Indeed, Kilian, Petrank and Rackoff [29] showed that no *black-box* simulation is possible for 4-round protocols and this was recently improved to 7-round [37].

and it is possible to perform forced opening. Note that this stands in contrast to the impossibility result even in the standard (non-concurrent) model [27]. They were also able to construct a three-round zero-knowledge protocol in the *resettable* setting [12], where prover is executed by a device that has no independent source of randomness and cannot record history.

## 6 Lower bound on the number of rounds for contract signing

We now show that any protocol for signing contracts must take a number of rounds linearly proportional to the *advantage* one side has over the other. While contract signing has been investigated extensively we haven't quite found a similar statement in the literature. The closest that we are aware of is Cleve's [15] lower bound regarding *gradual disclosure of secrets*, a task that can be used for contract signing. However, it does not directly imply lower bounds for contract signing.

Let $A_0$ and $A_1$ be the two parties as in Section 4.1. The key to understanding the limitations of contract signing algorithms is to have a *single*-dimensional notion of progress. We choose (computing time)/(prob. of success) as the measure of progress, though there are other reasonable possibilities.

This gives us the definition of unfairness. Fix a model of computation (say a specific Turing Machine, or, for nonuniform results, Boolean circuits over the complete $\{0,1\}^2 \mapsto \{0,1\}$ basis). Fix an adversary $\mathcal{A}$ that produces a contract $C$, plays the role of $A_{1-b}$ and attempts to come up with $S_b$. Let $\gamma(\mathcal{A})$ be the running time of $\mathcal{A}$ over its probability of success. Similarly, for $\mathcal{A}$ let $\delta(\mathcal{A})$ be the running time of the forced signature opening algorithm $R_b$ over probability that $R_b$ succeeds in finding an accepting $S_{1-b}$.

**Definition 2.** *The unfairness of a protocol is the worst case of all adversaries $\mathcal{A}$ of the quantity $\gamma(\mathcal{A})/\delta(\mathcal{A})$.*

*Remark 2.* Note that this notion of unfairness is more forgiving then the one in Section 4.1, in the sense that it ignores a case where $R_b$ retrieves the signatures in the same amount of time as $\mathcal{A}$, but with slightly smaller (but not negligible) probability. Nevertheless our lower bound is applicable to this definition.

**Definition 3.** *For every signing protocol we say that the protocol has* security gap $W$ *if the ratio between the running time an adversary needs in order forge a signature on a message without the signers agreement and the running time of the signing and verification algorithm is at least $W$.*

For a protocol to be useful the security gap must be large, say at least $2^{50}$.

**Theorem 2.** *For every contract signing protocol and forced opening algorithm, if the protocol consists of $k$ rounds and has security gap $W$, then the unfairness of the protocol is at least $W^{1/k}$.*

All our adversary $\mathcal{A}$ will do is to stop early. For each round $i$ of the protocol we consider all programs that get the transcript of the first $i$ rounds plus the secret of one of the sides and attempt to produce a valid signature on the contract. For such a program we are interested in its running time divided by the probability it succeeds in producing a valid signature where the probability is over the coin flips of all the participants. We assume that none was cheating except for early withdrawal.

Let $T_i$ be the minimum over all machines of the above product. If the contract signing protocol is secure at all, then $T_0$ should be large (super-polynomial). From completeness (i.e. that the protocol ends with valid signatures if the participants follow the protocol) for a $k$-round protocol $T_k$ should be small and by the assumption on the security gap we have $T_0/T_k \geq W$.

If $k$ is a constant, then there must be a large gap between $T_{j-1}$ and $T_j$ for some $1 < j \leq k$, more specifically

$$T_0 = T_k \cdot \frac{T_{k-1}}{T_k} \cdot \frac{T_{k-2}}{T_{k-1}} \cdots \frac{T_0}{T_1}.$$

Therefore for at least one $1 \leq j \leq k$ we have $\frac{T_{j-1}}{T_j} \geq (\frac{T_0}{T_k})^{1/k} \geq W^{1/k}$. This gives a way for one of the participants to make the protocol unfair - the one who receives a message at step $j$ stops afterwards and tries to create the signature. For this party the product of the time and probability of success is $T_j$, whereas for the other party it is at most $T_{j-1}$. Hence the unfairness of this protocol is at least $W^{1/k}$.

Note that if $W$ is at least $2^{50}$ and we want the unfairness to be at most 2, then the number of rounds must be at least 50.

This lower bound is applicable to a wider model than had been considered previously, In particular it applies to scenarios where the participants are timed (by real clocks) and their partner expect a response with in a certain amount of time (e.g. the model of [23]). In contrast, as we have seen in Section 5, the problem of collective coin-flipping has a constant round protocol in a timed model, as opposed to the untimed one.

## 7 Conclusions

We introduced the concepts of *timed commitments* and *timed signatures*, which are useful for contract signing, auctions and other applications. In auctions timed commitments ensure that users cannot make a sealed bid and then refuse to open the bid. We emphasized the importance of defending against parallel attacks in all applications. Our constructions for timed commitments and timed signatures resist parallel attacks by relying on modular exponentiation which is believed to be an inherently serial operation.

It would be interesting to try to construct timed commitments and timed signatures based on other primitives. For example, all lattice basis reduction algorithms are sequential. Can one build timed primitives based on lattice basis reductions? It is also interesting to see what other areas in cryptography can benefit from the ability to delay one's capabilities by a fixed time period.

## Acknowledgments

We thank Vitali Shmatikov and John Mitchell for discussions regarding abuse freeness that motivated this work. We thank Victor Shoup for many helpful comments and corrections.

The first author is supported by NSF and a grant from the Packard foundation. This work was done while the second author was visiting Stanford University and IBM Almaden Research Center. Partly supported by DOD Muri grant administered by ONR and DARPA contract F30602-99-1-0530.

## References

1. L. Adleman, and K. Kompella, "Using smoothness to achieve parallelism", proc. of STOC 1988, pp. 528–538.
2. N. Asokan, V. Shoup and M. Waidner, "Optimistic fair exchange of digital signatures", in proc. Eurocrypt'98, pp. 591-606, 1998.
3. G. Ateniese, "Efficient protocols for verifiable encryption and fair exchange of digital signatures", in proc. of the 6th ACM CCS, 1999.
4. M. Bellare and S. Goldwasser, "Verifiable Partial Key Escrow", in proc. of ACM CCS, pp. 78–91, 1997.
5. M. Bellare and S. Goldwasser, "Encapsulated key escrow". MIT Laboratory for Computer Science Technical Report 688, April 1996.
6. M. Bellare and P. Rogaway, "The Exact Security of Digital Signatures - How to Sign with RSA and Rabin", EUROCRYPT 1996, pp. 399-416
7. M. Ben-Or, O. Goldreich, S. Micali and R. L. Rivest, "A Fair Protocol for Signing Contracts", IEEE Transactions on Information Theory 36/1 (1990) 40-46
8. M. Blum, "How to Exchange (Secret) Keys", STOC 1983: 440-447 and ACM TOCS 1(2): 175-193 (1983)
9. L. Blum, M. Blum and M. Shub, A Simple Unpredictable Pseudo-Random Number Generator. SIAM J. Comput. 15(2): 364-383 (1986).
10. J. Y. Cai, R. J. Lipton, R. Sedgwick, A. C. Yao, "Towards uncheatable benchmarks, Structures in Complexity", in proc. Structures in Complexity, pp. 2–11, 1993.
11. J. Camenisch, M. Michels, "Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes", EUROCRYPT 1999, pp. 107-122.
12. R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. "Resettable Zero-Knowledge", ECCC Report TR99-042, Oct 27, 1999 and STOC 2000.
13. D. Chaum, and T. Pederson, "Wallet databases with observers", in Proceedings of Crypto '92, 1992, pp. 89–105.
14. R. Cleve, "Limits on the Security of Coin Flips when Half the Processors Are Faulty", in proc. STOC 1986, pp. 364–369.
15. R. Cleve, "Controlled gradual disclosure schemes for random bits and their applications", in proc. Crypto'89, 1990, pp. 573–588.
16. R. Cleve, R. Impagliazzo, "Martingales, collective coin flipping and discrete control processes", manuscript, 1993. Available: `http://www.cpsc.ucalgary.ca/~cleve/papers.html`

17. I. Damgård, “Concurrent Zero-Knowledge in the auxilary string model”, in proc. EUROCRYPT ’2000.
18. I. Damgård, “Practical and Provably Secure Release of a Secret and Exchange of Signatures,” J. of Cryptology 8(4): 201-222 (1995)
19. D. Dolev, C. Dwork and M. Naor, “Non-malleable Cryptography”, Preliminary version: Proc. 21st STOC, 1991. Full version: to appear, Siam J. on Computing. Available: `http://www.wisdom.weizmann.ac.il/~naor`
20. C. Dwork and M. Naor, “Pricing via Processing -or- Combatting Junk Mail”, Advances in Cryptology – CRYPTO’92, pp. 139–147.
21. C. Dwork and M. Naor, “An Efficient Existentially Unforgeable Signature Scheme and Its Applications”, J. of Cryptology 11, 1998, pp. 187–208,
22. C. Dwork and M. Naor, “Zaps and their applications”, manuscript.
23. C. Dwork, M. Naor and A. Sahai, “Concurrent Zero-Knowledge”, STOC, 1998.
24. S. Even, O. Goldreich and A. Lempel, “A Randomized Protocol for Signing Contracts,” CACM 28(6): 637-647 (1985).
25. A. Fiat and A. Shamir, “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”, CRYPTO’86, pp. 186–194.
26. J. A. Garay, M. Jakobsson and P. D. MacKenzie: “Abuse-Free Optimistic Contract Signing,” CRYPTO 1999, pp. 449-466.
27. O. Goldreich and H. Krawczyk. “On the Composition of Zero Knowledge Proof Systems,” SIAM J. on Computing, Vol. 25, No. 1, pp. 169–192, 1996.
28. D. Goldschlag and S. Stubblebine. “Publicly Verifiable Lotteries: Applications of Delaying Functions, Financial Cryptography,” 1998, LNCS Series, Springer-Verlag, 1998.
29. J. Kilian, E. Petrank and C. Rackoff, “Lower Bounds for Zero Knowledge on the Internet”, FOCS 1998, pp. 484–492.
30. M. Luby, S. Micali and C. Rackoff, “How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin,” FOCS 1983, pp. 11-21
31. A. Menezes, P. van Oorschot and S. Vanstone, “Handbook of Applied Cryptography,” CRC Press, 1996.
32. M. Naor, B. Pinkas and R. Sumner, “Privacy Preserving Auctions and Mechanism Design,” Proc. of the 1st ACM conference on E-Commerce, November 1999, pp. 129 – 139.
33. B. Pfitzmann, M. Schunter, M. Waidner, “Optimal Efficiency of Optimistic Contract Signing,” PODC 1998: 113-122
34. M. O. Rabin, “Transaction Protection by Beacons,” JCSS 27(2): 256-267 (1983)
35. R. Richardson and J. Kilian, “On the Concurrent Composition of Zero-Knowledge Proofs,” EUROCRYPT ’99, pp. 415–431, 1999.
36. R. Rivest, A. Shamir and D. Wagner, “Time lock puzzles and timed release cryptography,” Technical report, MIT/LCS/TR-684
37. A. Rosen, “A note on the round-complexity of concurrent zero-knowledge”, these proceedings.
38. V. Shmatikov and J. C. Mitchell, “Analysis of Abuse-Free Contract Signing,” in proc. of 4th Annual Conference on Financial Cryptography, 2000.
39. J. P. Sorenson, “A Sublinear-Time Parallel Algorithm for Integer Modular Exponentiation,” manuscript.
40. P. Syverson, ”Weakly Secret Bit Commitment: Applications to Lotteries and Fair Exchange”, Proceedings of the 1998 IEEE Computer Security Foundations Workshop (CSFW11), June 1998.