# Time-Lapse Cryptography

Technical Report TR-22-06

Michael O. Rabin* Christopher Thorpe

Harvard University
School of Engineering and Applied Sciences
Cambridge MA 02138
rabin@deas.harvard.edu cat@eecs.harvard.edu

20 December 2006

**Abstract**

The notion of "sending a secret message to the future" has been around for over a decade. Despite this, no solution to this problem is in common use, or even attained widespread acceptance as a fundamental cryptographic primitive. We name, construct and specify an implementation for this new cryptographic primitive, "Time-Lapse Cryptography", with which a sender can encrypt a message so that it is guaranteed to be revealed at an exact moment in the future, even if this revelation turns out to be undesirable to the sender. Our solution combines new ideas with Pedersen distributed key generation, Feldman verifiable threshold secret sharing, and ElGamal encryption, all of which rest upon the single, broadly accepted Decisional Diffie-Hellman assumption. We develop a Time-Lapse Cryptography Service ("the Service") based on a network of parties who jointly perform the service. The protocol is practical and secure: at a given time $T$ the Service publishes a public key so that anyone can use it, even anonymously. Senders encrypt their messages with this public key whose private key is not known to anyone – not even a trusted third party – until a predefined and specific future time $T + \delta$, at which point the private key is constructed and published. At or after that time, anyone can decrypt the ciphertext using this private key. The Service is envisioned as a public utility publishing a continuous stream of encryption keys and subsequent corresponding time-lapse decryption keys. We complement our theoretical foundation with descriptions of specific attacks and defenses, and describe important applications of our service in sealed bid auctions, insider stock sales, clinical trials, and electronic voting.

*Supported in part by National Science Foundation grant CNS-0205423.

## 1 Introduction and Related Work

First proposed by Timothy May [14], many attractive protocols have been proposed to encrypt messages to send into the future, usually under a name like "timed-release cryptography". We coin the phrase "Time-Lapse Cryptography" to distinguish protocols like ours, in which a fixed amount of time elapses between the ability to send a message (encrypt) and retrieve it (decrypt), from other methods in which only estimates of or lower bounds on elapsed time can be given.

### 1.1 Setting and Objectives

The setting for our service is as follows: At time $T$, Alice wishes to send Bob a message $m$ so that Bob may decrypt it only at or after a specified future time $(T + \delta)$. This decryption will be possible without any further action by Alice.

Our "Time-Lapse Cryptography Service" ("the Service") makes this possible. At or before time $T$, the Service publishes a public key $PK$ along with a statement that its corresponding private key $DK$ will be revealed at time $T + \delta$. Alice uses $PK$ to encrypt $m$ with random help $r$ using a probabilistic encryption scheme and sends the ciphertext $c = E_{PK}(m, r)$ to Bob. She is now committed to the content of the message, although Bob cannot yet see it. At time $(T + \delta)$, the Service reconstructs and publishes $DK$, which Bob obtains and uses to decrypt $c$ and recover $m$. (Of course, Alice, if she so wishes, can always reveal $m$ early by sending Bob $m$ and $r$.)

The primary objectives of our Service are as follows:

- The Service publishes a public key $PK$ associated with a start time $T$, duration $\delta$. It includes authenticating information with which users can unequivocally determine the authenticity of $PK$, $T$, and $\delta$.
- The private key $DK$ corresponding to $PK$ remains completely secret until time $T + \delta$.
- At time[1] $T + \delta$ the Service publishes the decryption key $DK$, along with authenticating information that allows any user to unequivocally determine the authenticity of $DK$.
- The Service is resistant to attacks that attempt to generate insecure public keys, prevent the generation of public keys, reconstruct the private keys early, or prevent accurate and timely reconstruction of private keys.

### 1.2 Summary of Contributions

We offer a complete description of a service and associated protocols that enable Time-Lapse Cryptography as described in Section 1.1. The Service we describe

[1]Plus a negligible delay $\epsilon$ for reconstruction described in Section 3.1.

is simple and easy to understand by anyone with an elementary cryptography background.

It is anonymous: the Service knows nothing about who might be using it; this increases privacy and eliminates any incentive for early private key reconstruction if the Service were to know a key were used for an important purpose.

The Service allows the originator of a message complete control over when the recipient may decrypt it, while guaranteeing that the recipient may decrypt the message at a specific future time.

The protocols rely only on well-studied and widely accepted cryptographic primitives: Pedersen distributed key generation (DKG) [18], Feldman verifiable secret sharing (VSS) [9], and the ElGamal cryptosystem [8].[2] Conveniently, the security of all three of these primitives rests on the widely believed assumption of the hardness of the Decisional Diffie-Hellman problem. This offers an elegant consistency and simplicity to the security of our proposal.

Our protocols guard against such attacks as: the Service being able to prematurely reveal the decryption key; the Service refusing to reconstruct the decryption key at the required time; users of the Service getting inconsistent views of the stream of public and private keys. We detail these and other attacks in Section 2.2. It will be clear from our construction that all these attacks are rendered impossible under generally accepted assumptions.

Our work also names and describes this protocol as a new cryptographic primitive that may be useful in complex protocols. This primitive can be viewed as a simple cryptographic commitment that is concealing *and* that cannot be repudiated. Alice is not only bound to not change content of the message; unlike in some other commitment schemes, such as those based on cryptographic hash functions, Alice furthermore may not prevent the message from being read by refusing to reveal the message (input to the hash function). When a binding commitment is required, Alice's digital signature on the ciphertext of a time-lapse encrypted message yields a commitment binding Alice to the still-secret content of the message.

An additional contribution of our work is a detailed enough description that will serve as a basis for an implementation of a time-lapse cryptography service, including details of and defenses against real-world attacks. We plan to deploy such an implementation in the coming months.

## 1.3 Applications

While we do not attempt to anticipate all of the possible applications that may be discovered for such a service, many useful applications already motivate its creation.

**Bids in sealed-bid auctions.** Our original motivation for this work came from joint work with David Parkes and Stuart Shieber on cryptographic auctions [17]. In our auction protocol, we realized the need for bidders to issue

[2]As described later, we recommend the use of more recent variants of these DKG and VSS protocols to eliminate specific attacks which may slightly bias the uniform distribution of the public keys.

commitments to their bids that were secret to even the auctioneer during the auction but could not be repudiated after the close of the auction. This prevents a type of abuse in which the auctioneer decrypts some bids and instructs favored bidders to refuse to unlock their bids (for example, because they offered far too much.)

Using our service, a bidder doubly encrypts her bids, first with the auctioneer's public key $PK_{AU}$ and then the public key $PK_S$ published by the time-lapse encryption service $S$. This creates the ciphertext $c = E_{PK_S}(E_{PK_{AU}}(\mathbf{Bid}))$, which is digitally signed by the bidder and published on a bulletin board. Thus no one, including the auctioneer, knows anything about her bid until either she reveals the random help value she used in $E_{PK_S}()$ or the appropriate amount of time elapses. No action of any bidder can prevent the auctioneer from decrypting her bid or the public from using her encrypted bid $E_{PK_{AU}}(\mathbf{Bid}))$ in verification protocols after the time-lapse expires.

**Insider stock trades.** An insider to a publicly-traded company could be legally obligated to issue advance commitments to stock transactions to mitigate the potential for abuse of inside information, as well as to protect the insider from false accusations of misuse of inside information. In certain circumstances, it is desirable that those commitments stay secret until shortly after the execution of the transaction in question. Clearly, a commitment that does not guarantee nonrepudiation does not suffice since an insider may publish in advance a concealed commitment to a trade and then refuse to reveal it in the event the trade is no longer desirable to him. If an insider encrypts his transaction in advance using a time-lapse cryptography service, he can always be legally compelled to complete the transaction although the details of the transaction remain secret until the appointed time. We suggest a protocol in which insiders issue their advance directives daily (say, for various lengths of time in advance) using the Service. These directives may be to buy, sell, or do nothing, which are indistinguishable under the semantic security of ElGamal. In this way an insider reveals no information to the market; while it is intuitive that this information could hurt the insider, some market microstructure research has shown that insiders can exploit disclosure rules due to the fact that "the market cannot observe whether an insider is trading on private information or for personal portfolio reasons [10]." Current SEC regulations require *ex post* disclosure for certain insiders, in part due to the argument that advance disclosure reveals too much information. The time-lapse cryptography Service answers this argument.

**Data collected in clinical trials.** In order to preserve the integrity of clinical trials, the data collected during such a trial may be encrypted using a time-lapse cryptography service. Because many of these trials are funded by companies who stand to make or lose significant amounts of money depending on their outcome, there is the potential for pressure to achieve a positive result. Use of our Service can mitigate this bias without revealing confidential information about the study before it is complete. Time-lapse cryptography prevents unethical scientists from cheating, and benefits ethical scientists by protecting them against false claims of fraud or pressure from their funders to achieve a particular outcome. Our protocol's property of early revelation also enables data collected in such trials to be revealed early in the case of necessity, for example, in cases that a drug is so effective it would be immoral not to offer it to the control group.

In one setting, scientists' data collection process uses our Service to encrypt data directly as they are being collected, for example, by diagnostic devices or computer user interfaces. The scientists would not be able to see the data collected until the conclusion of a phase of the study; this prevents observations of trends in early data collection from affecting future data collection practices.

In another setting, clinical data would be provided to the scientists in raw form immediately and to an auditing board encrypted via time-lapse cryptography. The scientists would preserve the confidentiality of their data during the study to prevent leaking of information by the auditing board, but would know that any tampering with results would be discovered after the expiration of the time-lapse.

**Electronic Voting.** In some voting applications, the publication of intermediate results may be undesirable, as it could unduly influence other voters or election officials. If votes are encrypted using time-lapse cryptography during an election, results can be kept completely confidential until polls close, as well as being assuredly revealed promptly when required.

## 1.4 Related Work

Solutions that do not have a fixed decryption time generally involve expensive sequential computations ("time-lock puzzles")[3] to recover an initial message, ensuring that the recipient cannot recover the data before some length of time, such as those proposed by Rivest et al. [19]. Other solutions that do not guarantee fixed time release are made possible by partial key escrow, first described by Bellare and Goldwasser in [2].

A number of ideas inspiring our approach use known encryption techniques in which the decryption key is kept secret until a fixed revelation time. Blake and Chan [3] describe the "Timed Release Encryption Problem" as a sender encrypting a message such that only a particular receiver can decrypt that message, and that only after a specific release time has passed, as verified by a single trusted, third-party time server. They solve this problem with a bilinear pairing on a Gap Diffie-Hellman group, which requires reasonable cryptographic assumptions.

Blake and Chan's solution is similar to those employed in identity-based cryptography. Other work sharing this connection is the work by Cheon et al. [5] formalizing "secure timed-release public key encryption" and its equivalence to strongly key-insulated public key encryption. Their solution, also based on a bilinear map, requires a trusted "timed-release public server" that periodically publishes information, based on a private secret, that enables decryption of previously encrypted texts. Dodis and Yum [7] proposed a related protocol in which digital signatures become verifiable only at a fixed future time $t$ upon

[3]Merkle [16] is generally credited with inventing these "puzzles".

publication by a trusted third party of "some trapdoor information associated with the time $t$."

Our solution is also related to "token-controlled" public key encryption, first introduced by Baek et al. in [1]. In token-controlled encryption, messages are encrypted with both a public encryption key and a secret token, and can only be decrypted with the private decryption key after the token is released. Time-lapse cryptography and token-controlled encryption share many important applications, and in fact an approach similar to time-lapse crypto could be used as a means of securely generating and distributing the secret tokens with distributed trust.

Rivest et al. [19], in addition to time-lock puzzles, offer the first description we know of the use of a secret decryption key for time-lapse cryptography; in their scheme, a trusted third party creates and distributes public and private keys at appropriate times. Our protocol is similar, but replaces their trusted third party with a network of parties and an assumption that no fewer than a specified number of these parties need to be trusted.

Di Crescenzo et al. [6] employ a trusted time server and a new primitive called "conditional oblivious transfer" to send messages into the future where the server never learns the sender's identity. (It does learn the receiver's identity.)

## 2 Preliminaries and Assumptions

Our service consists of the following major components:

- A network of $n$ participating parties $P_1, \ldots, P_n$
- Distributed key generation of the public and private keys
- Verifiable threshold secret sharing of the private key
- Secure multi-party reconstruction of components of the private key
- Reconstruction and publication of the private key
- Secure public and private bulletin boards for posting of intermediate and final results

The protocol is conducted by a "Time-Lapse Cryptography Service" ("the Service") consisting of $n$ parties $P_1, \ldots, P_n$. The protocol allows for the possibility that these parties may only be intermittently available. It also allows for the existence of adversaries that may attempt to disrupt the protocol in various ways. Call the generation of a public key and the corresponding reconstruction of the private decryption key an "action" of the Service. We assume a threshold $t$ such that during any one action, at most $t-1$ parties may disrupt the protocol by revealing secret information, submitting false information, or refusing to participate in the action. Any such party will be informally referred to as being *improper*. We also assume that during the entire action, at least $t$ parties strictly follow the protocol. Such parties will be informally referred to as being *proper*. This implies that $n \geq 2t - 1$.

We postulate that for the ElGamal encryption, there is a publicly agreed-upon cyclic group $G$ and generator $g \in G$ of prime order $q$. For this work we assume that $2q + 1$ is a prime $p$, and that $G$ is the set of quadratic residues modulo $p$; hence, all elements of $G$ other than $\{1, -1\}$ have order $q$. This ensures semantic security vis-à-vis quadratic residuosity.

We further assume that $p$ and $q$ are selected with appropriate attention to cryptanalysis, so that the encryption scheme used is resistant to known attacks involving vulnerabilities of particular "unsafe primes." Without loss of generality, we will refer to only one group $G$ and public generator $g$ for both ElGamal encryption and the verification of shared secrets. Other groups $G$ are possible, most notably elliptic curve groups that offer improved efficiency.

## 2.1 Implementation Considerations

The Service will be implemented on a network of autonomous computers, each of which represents a party $P_i$ in our protocol. Each party follows the protocol described below; it obtains the schedule of public key generation and private key reconstruction from a set of manager computers we next describe.

For further efficiency, reliability and resistance to attacks, we employ a small network $M$ of $K$ managers that act as a "managing team" for the Service. The role of the managing team is to create the schedule of the public and corresponding private keys to be produced by the Service; to maintain an internal bulletin board for use by the parties comprising the Service; and to maintain a public bulletin board for users of the Service. Integrity of these bulletin boards is achieved by each manager maintaining his own copies of these two bulletin boards. Parties and users will look at messages posted on each of the managers' copies of the bulletin boards and determine the correct values by a majority of postings.

The authoritative time for all actions shall come from an assumed universally accessible clock (Section 2.4), and no party or manager shall rely on an internal clock. All computers comprising the Service should be maintained by administrators with experience in security considerations and running operating systems with up-to-date security patches.

## 2.2 Resistance to Attacks

Up to $t - 1$ improper parties $P_i$ may attack the Service in various ways. We describe in detail our protocol's resistance to the following attacks by these improper parties at the appropriate phase of our protocol in Section 4.

- Sabotaging the joint construction of a valid, random $PK$
- Posting an incorrect value of $PK$
- Prematurely reconstructing $DK$ (prior to time $T + \delta$)

- Sabotaging the reconstruction of $DK$ at time $T + \delta$

In addition, an improper party can attack the distributed key generation algorithm we describe by introducing a slight bias into the distribution of possible public keys. We refer the reader to work by Gennaro et al. [12] for a complete description of this attack and a modified algorithm that prevents it. Those implementing the protocol may wish to periodically review cryptology research on distributed key generation in order to guard against new attacks.

We also point out that improper parties or users of the Service may mount denial of service attacks by attempting to overload the Service with internal or public bulletin board postings or requests for keys. The managers of the Service can prevent such attacks by appropriate rationing of postings and requests. Of course, there exist other known possible denial of service attacks, and corresponding countermeasures, that are outside the scope of this work.

## 2.3 Security Assumptions

The protocol employs the ElGamal encryption scheme [8]. ElGamal's scheme *is* semantically secure under chosen plaintext attacks (CPA): adversaries can encrypt as many messages as they want and gain no information about the private key or any other encrypted message. ElGamal is known to be trivially malleable and hence insecure under chosen ciphertext attacks (CCA-1). We do not view this as a security risk, because no ciphertexts can be decrypted with the private key before its reconstruction and publication, and it is expected at that time that all ciphertexts encrypted with that key can be decoded by anyone. Malleability is not of concern in our protocol, because it can be avoided by signing encrypted messages via an appropriate, nonmalleable digital signature scheme.

We assume that each party $P_i$ uses a computer that accurately and secretly performs the computations we describe and securely stores all $P_i$'s secret data. We assume the parties back up data in some secure way for disaster recovery, though it must be a method that makes stealing the secrets from backups at least as difficult as compromising the hosts themselves.

## 2.4 Communications Assumptions

We assume that each party $P_i$ can communicate privately and secretly with any other party $P_j$. For example, each party may have a public/private cryptographic key pair and all parties will know every other party's public key.

In addition, our protocol will require posting of various intermediate steps and results. The parties will employ the internal bulletin board provided by the managing team for that purpose, as described in Section 2.1. Posting of any message $m$ by a party $P_i$ will always be accompanied by $P_i$'s digital signature $SIGN_i(m)$.

We also assume a universally accessible and tamper-resistant clock, such as provided by the US NIST, that determines times for actions taken by the Service.

### 2.5 Summary of ElGamal Encryption

As described above, we assume a publicly known group $G$ and generator thereof $g$. The Service creates and publishes an ElGamal public key $PK = g^x$ as described later; the private key is $DK = x$.

To encrypt a message $m$, Alice first obtains the public key $PK = g^x$ and creates a random help value $y \xleftarrow{R} [1, q-1]$. She then computes the ciphertext $c$ as a pair: $c = (g^y \pmod{p}, m \cdot g^{xy} \pmod{p})$.

Alice then sends this pair $c$ to Bob. By elementary algebra, Bob can recover $m$ when the Service publishes the private key $x$ or Alice later sends him the random help value $y$.

## 3 How the Service Works

For a less formal introduction to our protocol, we recall the reader to our high-level overview in Section 1.1.

### 3.1 What the Service Does

The Service creates, publishes and maintains "time-lapse cryptographic key structures" that represent public time-lapse cryptography keys with a specific lifetime. The Service may generate these structures on a periodic basis for public convenience; for example, each day it might release keys with a lifetime of 1 week, or every 30 minutes release keys with a lifetime of 2 hours. These schedules are posted by the managers to the public bulletin board. In addition, the Service can accept requests from clients to generate new keys with a particular lifetime; the managers accept these requests and post them on the public bulletin board. The parties $P_i$ construct the key structures according to the protocol, individually sign them, and publish the signed key structures on the public bulletin board.

For each key required by convention or client request, the Service will generate a key structure $K_{ID} = (ID, T_{ID}, \delta_{ID}, PK_{ID})$ consisting of a unique identifier $ID$, a publication time $T_{ID}$, a "time-lapse" $\delta_{ID}$, and a public key $PK_{ID}$. Each party $P_i$ publishes the key structure and signature thereof $(K_{ID}, SIGN_i(K_{ID})$ on the public bulletin board.

At time $(T_{ID} + \delta_{ID})$ the Service will reconstruct and publish the associated private key $DK_{ID}$. The public key and private key for $K_{ID}$ are related by the equation $PK_{ID} \equiv g^{DK_{ID}} \pmod{p}$. It is assumed that $g$ is public. It is crucial that the private key $DK_{ID}$ is known to no one, and never reconstructed, before the appropriate time. As before, each party $P_i$ publishes the reconstructed private key and signature thereof $(DK_{ID}, SIGN_i(DK_{ID})$ on the public bulletin board.

There is a subtle issue in that reconstruction of the private key is not in fact instantaneous. In practice, the Service will begin reconstruction of the private key $DK_{ID}$ at time $(T_{ID} + \delta_{ID})$ and publish $DK_{ID}$ at time $(T_{ID} + \delta_{ID} + \epsilon)$ where $\epsilon$ is the time required to reconstruct the private key. We assume that $\epsilon$ can be made negligible in comparison to any time-lapse $\delta_{ID}$ and will be on the order of a fraction of a second, and therefore we generally assume $\epsilon = 0$ for convenience. At the beginning of the time lapse, we assume that the time $T_{ID}$ is an upper bound on the time when the key is released, and that the Service may release a key required at time $T_{ID}$ at any time at or before $T_{ID}$.

## 3.2 What the Clients Do

When Alice wishes to send Bob a message $m$, she requests or selects an appropriate key structure $K_{ID}$ from the Service. Alice does not need to identify herself in any way in order to do this; because the Service publishes the key structures on the public bulletin board, Alice may use any mechanism for obtaining the public key structure, e.g. a friend or an anonymous Web proxy server. Alice then verifies the published digital signatures $SIGN_i(K_{ID})$ match the published key structure $K_{ID}$ for a minimum of a threshold $t$ parties $P_i$, and that these parties' $K_{ID}$ are identical. This guarantees that $PK_{ID}$ is the public key generated by all the proper parties, and its corresponding decryption key $DK_{ID}$ will be subsequently reconstructed and correctly posted by all the proper parties.

To send the message, Alice encrypts $m$ using ElGamal encryption; she creates a random help value $y \overset{R}{\leftarrow} [1, q-1]$ and privately sends Bob the pair $c = (g^y \pmod p), m \cdot PK_{ID}^y \pmod p))$ as well as the index $ID$ of the key structure $K_{ID}$ whose public key she used. Alice may at this stage apply other appropriate cryptographic primitives, such as a digital signature or message authentication code, depending on the application. If Alice wishes to send a longer message than can be accommodated by the group $G$, she may use our protocol to encrypt and send a secret key for a block cipher and encrypt her actual message with that block cipher, or she may break her message up into smaller chunks and encrypt each one.

Alice now has no ability to stop Bob from decrypting her message. Bob receives $c$ and stores it, then waits for Alice to send $y$ or for time $(T_{ID} + \delta_{ID})$, whichever comes first. If Alice sends him $y$, he decrypts $m$ using $g^{PK_{ID}}$ and $y$; if she does not, he obtains $PK_{ID}$ from the Service and decrypts $m$ using that.

# 4 Protocol for the Parties $P_i$ in the Service

We use a standard distributed key generation (DKG) algorithm as described by Pedersen [18], and employ Paul Feldman's simple verifiable secret sharing (VSS) scheme [9] to guarantee the authenticity of the generated keys.[4]

Throughout this section we shall designate a set of "qualified" parties $Q$ which are the parties that have complied completely with the protocol and not been disqualified for any reason. It will turn out that for any action (i.e. the construction of an encryption key $PK$ and the subsequent reconstruction

[4]See Section 2.2 for a brief discussion of a subtle attack and a reference to a modified algorithm defending against it.

of the corresponding decryption key $DK$), $Q$ will include all proper parties. Consequently, $|Q| \geq t$ at all times.

## 4.1 Distributed key generation

Whenever a fixed "preparation interval" before a posted key generation time $T$ is reached, each party $P_i$ begins the protocol. For example, the Service might schedule a 1-week key to be released each day at 10:00 am Eastern Time; the parties begin preparing this key a few minutes ahead of schedule so that it can be released at or before 10 am. It will be seen later on that parties to the Service may be disqualified during the creation phase of the public key by demonstrably violating the protocol. We shall refer to the set of parties that were *not* disqualified as the set $Q$ of "qualified parties". It will turn out that all proper parties (and possibly some improper parties) $P_i$ will be members of $Q$, and that every proper party will have the same view of (value for) $Q$.

Each party $P_i$ should choose a random $x_i \overset{R}{\leftarrow} [1, q-1]$. This $x_i$ constitutes $P_i$'s candidate *component* of the private key. It will turn out that the private key will be $x = \sum_{i \in Q} x_i \pmod q$. Each $P_i$ should then compute $h_i = g^{x_i} \pmod p$ and post $(h_i, \mathit{SIGN}_i(h_i)$ on the internal bulletin board. It will turn out that the public key will be $h = \prod_{i \in Q} h_i \pmod p$. This $h_i$ is $P_i$'s candidate component of the public key. Any party $P_i$ who does not post $h_i$ is disqualified. Obviously, all proper parties will have the same view of which parties were disqualified at this point.

## 4.2 Sharing the private key components

In order to ensure that the private key $x$ corresponding to the public key $h$ will be correctly reconstructed at time $T + \delta$, we have to protect against the possibility that improper parties will refuse to reveal their component $x_i$ of the private key $x$ or reveal a false value instead of $x_i$. This is achieved by use of verifiable threshold secret sharing. During this phase, further parties $P_i$ may be disqualified.

Each party $P_i$ should create a random polynomial of degree $k = t - 1$ in $F_q[z]$:

$$f_i(z) = x_i + a_{1i}z + a_{2i}z^2 + \ldots + a_{ki}z^k$$

The secret key component is $f_i(0) = x_i$. Each party $P_i$ should compute secret shares $x_{ij} = f(j)$ and verification commitments $c_0 = h_i = g^{x_i}, c_1 = g^{a_{1i}}, \ldots, c_k = g^{a_{ki}}$. (All commitments $c_i$ are computed $\pmod p$.) Each $P_i$ then privately sends to all $P_j, j \in [1, n]$, $(j, x_{ij}, \mathit{SIGN}_i(j, x_{ij})$ and posts on the internal bulletin board signed commitments $(c_0, \mathit{SIGN}_i(c_0)), \ldots, (c_k, \mathit{SIGN}_i(c_k))$. Every $P_j$ can now verify that $x_{ij}$ is a correct share by checking $(*)$:

$$(*) \quad g^{x_{ij}} \equiv c_0 c_1^j c_2^{j^2} \ldots c_k^{j^k} \pmod p$$

(In our proposal, index $j$ is the argument to the polynomial for all $P_j$.)

At this point an improper $P_i$ can disrupt the process in one of two ways. First, he may send $P_j$ an incorrect share $x_{ij}$ of his component $x_i$. In this case, $P_j$ posts the triple $(j, x_{ij}, SIGN_i(j, x_{ij}))$ on the internal bulletin board. The proper parties will check whether $x_{ij}$ is valid according to $(*)$. If it is an invalid share, then $P_i$ is disqualified. All parties can check whether $x_{ij}$ is a valid share according to $(*)$. All proper parties will arrive at the same conclusion as to whether $P_i$ should be disqualified.

Second, $P_i$ may have failed to send $P_j$ the share $x_{ij}$. In this case $P_j$ posts a signed protest to the internal bulletin board. $P_i$ is then required to reveal $x_{ij}$ on the internal bulletin board by posting a signed message $(j, x_{ij}, SIGN_i(j, x_{ij}))$. Every party can then verify the posted share $x_{ij}$ according to $(*)$. If it is invalid, then $P_i$ is disqualified. Again, all proper parties will reach the same conclusion as to the disqualification of $P_i$.

Putting all the above together, it is clear that all proper parties now have the same view of the value $Q$, the set of qualified parties.

Despite the above posting of some shares, the secrecy of the private key is preserved until time $T+\delta$. Consider first shares $x_{ij}$ of the private key component $x_i$ of a proper party $P_i$. Only improper parties $P_j$ will (unjustly) demand revelation of such shares. Thus, just a total of at most $t-1$ shares of $x_i$ will be posted. By the properties of Shamir secret sharing [20], the component $x_i$ remains random to the improper parties, and any observer of the internal bulletin board. Of course, the improper parties can always circulate the shares they received anyway: an adversary gains nothing by this revelation. Next, consider shares $x_{ij}$ of an improper party $P_i$ who refuses to send $P_j$ its share. The posting of $P_i$'s shares may reveal $x_i$. However, even if every improper $P_i$ would broadcast its component $x_i$ of the private key $x$, the private key remains secret until the components $x_j$ of the proper parties are revealed and this happens only at time $T+\delta$.

## 4.3 Publishing the public key

Now, each qualified party $P_j$ holds the public key $h$, a component $x_j$ of the private key $x$, and shares $x_{ij}$ for all qualified parties $P_i$. These latter shares are kept for the reconstruction of any missing components $x_i$ that are unavailable at the conclusion of the protocol if $P_i$ is unavailable or corrupted.

Every qualified party $P_j, j \in Q$ forms $h = \prod_{i \in Q} g^{x_i} \pmod{p}$ and the key structure $K_{ID} = (ID, PK_{ID} = h, T_{ID}, \delta_{ID})$. and posts $(K_{ID}, SIGN_j(K_{ID}))$ on the internal *and* public bulletin boards. A simple analysis shows that all the parties proper during this action will post the same value for $K_{ID}$. The number of such proper parties strictly exceeds $n/2$. Consequently, any user viewing the public bulletin board can unambiguously extract a valid value for $K_{ID}$. The public key $PK_{ID}$ can now be used for time-lapse encryption. Clearly, users can and should verify the digital signatures on data posted on the public bulletin board.

## 4.4 Reconstructing and publishing the private key

At the appointed time ($T_{ID} + \delta_{ID}$) for the reconstruction of the private key $DK_{ID}$, by definition, all parties proper for this action will correctly participate. Consequently, at least $t$ proper parties will do so. Parties consult the public bulletin board maintained by the managers to obtain the list of reconstruction times, and begin the reconstruction protocol when the time $T_{ID} + \delta_{ID}$ for reconstructing $DK_{ID}$ is reached on the universal clock.

First, every party $P_i$ should publish its component $x_i$ of the private key $x = DK_{ID}$ to the internal bulletin board. By definition, all proper parties do so. Note that even after this is done, certain components $x_i$ previously provided by some $P_i \in Q$ may be missing if the party $P_i$ in question is in fact improper. Every proper party then checks that for every $P_i \in Q$, the posted $x_i$ satisfies the equation $g^{x_i} \equiv h_i \pmod{p}$, where $h_i$ is as published in the previous step. For any $P_i \in Q$ who has not posted $x_i$ or for whom this verification fails, the parties need to reconstruct the correct $x_i$. Obviously, by definition, at least the parties proper within this action will do so. Note that the parties $P_i \notin Q$ are of no interest since their candidate shares did not enter into the construction of the private key $x$.

Now, every party $P_j$ should post the $x_{ij}$ it received from $P_i$ during the distributed key generation phase described in Section 4.2.

Note that at this point, every proper party $P_j$ has either received a verified $x_{ij}$ from $P_i$ which it posts, or in the "Sharing the Private Key" phase (Section 4.2) of the protocol, demanded of $P_i$ to post to the internal bulletin board the share $x_{ij}$. Furthermore, that posted share was verified. This holds because otherwise $P_i$ would have been disqualified and not included in $Q$.

In summary, every proper $P_j$ now sees on the internal bulletin board at least $t$ valid shares $x_{ij}$ of $P_i$'s component $x_i$ of the private key $x = DK_{ID}$. The party $P_j$ uses any $t$ shares $x_{ij}$ to reconstruct $x_i$ by polynomial interpolation.

After this is done, every proper party $P_j$ has all the components $x_i$ for all the parties $P_i \in Q$. Every such $P_j$ now computes the sum $DK_{ID} = x = \sum_i x_i \pmod{q}$ and publishes $(ID, DK_{ID}, SIGN_j(ID, DK_{ID}))$ to the public bulletin board. Now, there will be strictly more than $n/2$ signed postings agreeing on the value of $DK_{ID}$. Consequently, any user looking up the value of $DK_{ID}$ can unequivocally determine it, even if improper parties attempt to sabotage the reconstruction or the posting of the private decryption key.

The *Handbook of Applied Cryptography* [15] offers a concise description of polynomial interpolation in Section 12.71 in its description of Shamir's $(t, n)$ threshold secret sharing scheme [20].

## 4.5 Proactive renewal of components and shares

Because there may be applications where a time-lapse cryptographic key has a very long life (for example, a year or more), it may be prudent to periodically redistribute the shares of each party's component of the private key and shares thereof for additional security. With such a system in place, an adversary has a limited time to obtain the required number of secret components before the components are renewed and past components are no longer useful. A protocol for doing so for ElGamal cryptosystems is described by Herzberg et al. [13] and related work. This enhancement can be directly combined with the present paper's protocol.

## 5 Conclusions and Future Work

We have developed a simple, practical and clear protocol that solves the problem of "sending a secret message into the future" and a Service that implements it. Our formal treatment firmly establishes this idea as a useful cryptographic primitive; previous work and our suggested applications demonstrate broad applicability. Our work goes beyond a purely theoretical foundation and describes how our Service might be implemented in practice with important practical details, including resistance to specific attacks. We have isolated the fundamental elements of the "Time-Lapse Cryptography" primitive in our construction. This allows for established primitives to perform additional cryptographic functions. For example, the sender Alice of a time-lapse encrypted secret to Bob can restrict subsequent revelation solely to Bob by further encrypting the ciphertext again with Bob's public key; she can achieve non-malleability via a message authentication code; she can apply her digital signature to prove she sent the message, etc. Thus we have a clean, independent primitive that is easy to understand and employ in more complex protocols.

Plans for future work include a complete implementation of our Service on a distributed network of computers made available for public use. During this process we will improve the details of our specification and deepen our understanding of the practical security of the underlying protocols we employ. Another extension to the approach we describe is its application to time-lapse cryptography in other cryptosystems using distributed generation of other cryptographic keys, most notably composites of two large primes as used in RSA and Paillier encryption. Boneh and Franklin first proposed an efficient distributed RSA key generation protocol [4]; Frankel et al. offer a more robust formulation in [11].

We also anticipate that we and others will invent and describe novel applications of this technology once it is publicly available. For example, the homomorphic properties of ElGamal and Paillier encryption may be useful in a time-lapse setting. We have also considered modified time-lapse cryptography protocols in which we retain the properties of sender anonymity and guaranteed future decryption if the sender does nothing, yet allow the sender to delay decryption until a later time upon request to the Service. One application of this extension is to the encryption of a will, in which the testator wishes to postpone its revelation until required.

## 6 Acknowledgments

This work was supported in part by National Science Foundation grant CCR-0205423 to Rabin. We thank David Mazières, David Parkes, and Stuart Shieber for helpful comments and discussions that contributed to this work.

## References

[1] J. Baek, R. Safavi-Naini, and W. Susilo. Token-controlled public key encryption. In R. H. Deng, F. Bao, H. Pang, and J. Zhou, editors, *Proceedings of ISPEC 2005*, volume 3439 of *Lecture Notes in Computer Science*, pages 386–397. Springer Verlag, 2005.

[2] M. Bellare and S. Goldwasser. Verifiable partial key escrow. In *ACM Conference on Computer and Communications Security*, pages 78–91, 1997.

[3] I. F. Blake and A. C.-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing.

[4] D. Boneh and M. K. Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology - CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 425–439. Springer Verlag, 1997.

[5] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Timed-release and key-insulated public key encryption. Cryptology ePrint Archive, Report 2004/231, 2004.

[6] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. *Lecture Notes in Computer Science*, 1592:74 ff., 1999.

[7] Y. Dodis and D. H. Yum. Time capsule signature.

[8] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, IT-31(4):469–472, 1985.

[9] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. *IEEE Symposium on Foundations of Computer Science*, pages 427–437, 1987.

[10] M. J. Fishman and K. M. Hagerty. The mandatory disclosure of trades and market liquidity. *The Review of Financial Studies*, 8(3):637 ff., 1995.

[11] Y. Frankel, P. D. MacKenzie, and M. Yung. Robust efficient distributed RSA-key generation. In *Proceedings of the seventeenth annual ACM symposium on Principles of Distributed Computing*, page 320, 1998.

[12] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Lecture Notes in Computer Science*, 1592:295 ff., 1999.

[13] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *ACM Conference on Computer and Communications Security*, pages 100–110, 1997.

[14] T. C. May. Timed-release crypto. In *The Cyphernomicon: Cypherpunks FAQ and More, v. 0.666*, chapter 14.5. September 1994.

[15] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[16] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, Apr. 1978.

[17] D. C. Parkes, M. O. Rabin, S. M. Shieber, and C. A. Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. In *ICEC '06: Proceedings of the 8th international conference on Electronic commerce*, pages 70–81, New York, NY, USA, 2006. ACM Press.

[18] T. P. Pedersen. Non-interactive and informationtheoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO*, Lecture Notes in Computer Science, pages 129–140. Springer Verlag, 1991.

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

[20] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.