# Optimistic Fair Exchange of Digital Signatures*

N. Asokan[†] Victor Shoup[‡] Michael Waidner[‡]

October 18, 1999

**Abstract**

We present a new protocol that allows two players to exchange digital signatures over the Internet in a fair way, so that either each player gets the other's signature, or neither player does. The obvious application is where the signatures represent items of value, for example, an electronic check or airline ticket. The protocol can also be adapted to exchange encrypted data. It relies on a trusted third party, but is "optimistic," in that the third party is only needed in cases where one player crashes or attempts to cheat. A key feature of our protocol is that a player can always force a timely and fair termination, without the cooperation of the other player, even in a completely asynchronous network. A specialization of our protocol can be used for contract signing; this specialization is not only more efficient, but also has the important property that the third party can be held *accountable* for its actions: if it ever cheats, this can be detected and proven.

*This is an extensively revised and expanded version of an extended abstract in *Eurocrypt '99*, and of IBM Research Report RZ 2973 (Dec. 17, 1997).

[†]Communications Systems Laboratory, Nokia Research Center, P.O. Box 407, FIN-00045 Finland, e-mail: `n.asokan@nokia.com`; work done while at IBM Research, Zürich Research Laboratory

[‡]IBM Research, Zürich Research Laboratory, CH-8803 Rüschlikon; e-mail: {`sho,wmi`}`@zurich.ibm.com`

# 1 Introduction

As more business is conducted over the Internet, the *fair exchange problem* assumes increasing importance. For example, suppose player $A$ is willing to give an electronic check to player $B$ in exchange for an electronic airline ticket. The problem is this: how can $A$ and $B$ exchange these items so that either each player gets the other's item, or neither player does.

Both electronic checks and electronic airline tickets are implemented as digital signatures. Presumably, many other items to be exchanged over the Internet will be so implemented. Therefore, it seems fruitful to focus our attention on the fair exchange of digital signatures.

Of course, one could use an on-line trusted third party in every transaction to act as a mediator: each player sends his item to the third party, who upon verifying the correctness of both items, forwards the item to the other player. This is a rather straightforward solution; variations are discussed in the papers [CTS95, DGLW96, FR97].

In this paper, we present a new protocol for fair exchange that takes a different approach. Our protocol uses a trusted third party, but only in a very limited fashion: the third party is only needed in cases where one player attempts to cheat or simply crashes; therefore, in the vast majority of transactions, the third party will not need to be involved at all. Following [ASW97], we call our protocol *optimistic*; in addition to [ASW97], optimistic protocols for several variants of the fair exchange problem are discussed in [BDM98, BP90, Mic97].

Compared to a protocol using an on-line third party, the optimistic approach greatly reduces the load on the third party, which in turn reduces the cost and insecurity involved in replicating the service in order to maintain availability. It also makes it more feasible to implement the trusted third party service as a distributed, fault-tolerant system, eliminating the single point of failure.

Our new protocol can be used to exchange commonly used digital signatures, including RSA [RSA78], DSS [Kra93], Schnorr [Sch91], Fiat-Shamir [FS87], GQ [GQ90], and Ong-Schnorr [OS91] signatures, as well as the payment transcripts used in Brands' [Bra93] off-line, anonymous cash scheme. Moreover, the protocol can also be adapted to exchange digital content, such as music or stock quotes, and to the related problem of certified e-mail.

Our protocol also enjoys the following properties:

(1) It works in an asynchronous communication model: there is no need for synchronized clocks, and one player cannot force the other to wait for any length of time—a fair and timely termination can always be forced by contacting the third party.

(2) To use it, one need not modify the signature scheme or message format *at all*. Thus, it will inter-operate with existing or proposed schemes for electronic checks, coins, tickets, receipts, etc., without *any* modification to these schemes.

(3) The protocol does not require the players to "pre-register," or otherwise interact in advance with the trusted third party.

(4) It is practical. A typical exchange requires only a few rounds of interaction, transmission of a few KBytes of data, and a couple thousand modular multiplications.

(5) The protocol can be proved secure (modulo standard intractability assumptions) in the random hash function model [BR93], where a hash function is treated *as if* it were a "black box" that contains a random function.

(6) The two players need not sacrifice their privacy in making use of the trusted third party.

We wish to emphasize the importance of property (1). Previous optimistic protocols for fair exchange could easily leave one player "hanging" for a long time, without knowing if the exchange was going to complete, and without being able to do anything about it. Not only can this be a great inconvenience, it can also lead to a real loss in the case of time-sensitive data like stock quotes. In our protocol, this cannot happen so long as the third party is available. Clearly identifying this problem and providing an effective solution is probably the most important contribution of this paper.

We stress the practical importance of property (2): it allows a general-purpose fair exchange service to be deployed without the cooperation of the institutions responsible for the items being exchanged (banks, airlines, etc.). Indeed, it seems quite unrealistic to expect these institutions to redesign their schemes and all of the relevant software to accommodate a fair exchange protocol if this has not already been designed for. Our protocol can accommodate any common signature scheme without modification. Previous optimistic protocols for fair exchange do not allow for this: these protocols either require that the item being exchanged have a special structure to facilitate the exchange protocol, or they partially sacrifice fairness, with one player ending up with just an affidavit from the third party that the other player owes him something. In our protocol, the two players get the real thing—not a substitute or affidavit.

We also point out that in practice, the most common threat to a fair exchange is not malicious behavior by a player, but simply the possibility that one player crashes in the middle of the exchange. Our protocol deals equally well with both types of threats.

In addition to the above general protocol for fair exchange of digital signatures, we give a specialized version of this protocol for contract signing. Here, we allow ourselves the additional flexibility of defining the form of a valid contract (sacrificing property (2) above). The protocol still works in an asynchronous communication model, is much more efficient than the general protocol, and enjoys the additional important property of *accountability:* if the trusted third party ever cheats, then this can be detected and proven. Thus, the third party can be held accountable for its actions, and perhaps be sued should it misbehave.

## 2 Technical Overview

In this section, we discuss some of the more subtle issues that arise in the design of an optimistic fair exchange protocol, and give a high-level sketch of our new protocol.

In §2.1, we discuss the key cryptographic tool one needs to build an optimistic fair exchange protocol—verifiable escrow. In §2.2, we discuss a simple, but flawed optimistic fair exchange protocol. In §2.3, we sketch our new optimistic fair exchange protocol. In §2.4, we sketch our security model and proof techniques. In §2.5, we outline the remainder of the paper.

### 2.1 Escrow and verifiable escrow schemes

A key idea behind our protocol is to use the trusted third party as an "escrow service." Essentially, a player can send an encryption of his signature to the other player, where the signature is encrypted under the public key of the trusted third party so that the recipient of the encryption can have it decrypted by the trusted third party, if necessary.

In any such escrow scheme, one critical—but usually overlooked—point is that the creator of the encryption should have some ability to control the conditions under which the encryption will be decrypted by the trusted third party. That is, we need a cryptographic *mechanism* with which we can implement a decryption *policy*.

To this end, we define an *escrow scheme* as a public key encryption scheme in which an encryption comes with an attached *condition* that can be interpreted by the trusted third party to enforce some sort of decryption policy. It should be infeasible to modify the condition attached to an escrow; otherwise, the decryption policy could be thwarted. Such an escrow scheme can be easily implemented using any public key cryptosystem secure against chosen ciphertext attack. All of this will be explained in greater detail later.

In addition to such a basic escrow scheme, we will need an escrow scheme that is *verifiable*, in the sense that the player who receives this escrow can verify that it is indeed the escrow of a signature of the desired form with a correct condition attached.

Although one can implement such a verifiable escrow scheme using general zero-knowledge proof techniques [GMW91, BCC88], the resulting schemes would not be very practical. We show how to obtain practical verifiable escrow schemes for a broad range of signature schemes. To do this, suppose a player has "promised" a signature in exchange for another signature; then, we first have this player reduce the "promise" of a signature to a "promise" of a particular homomorphic pre-image (either a discrete logarithm or RSA pre-image); this reduction should not make it any easier to compute the signature, while simultaneously guaranteeing that the promised signature can be recovered from the promised pre-image. Second, we use a standard "cut-and-choose" interactive proof, in combination with an arbitrary, ordinary escrow scheme, to verifiably escrow the relevant homomorphic pre-image.

Our method to verifiably escrow a homomorphic pre-image is very similar to a scheme designed by Bellare and Goldwasser [BG96].

Stadler [Sta96] gives a different method for verifiably encrypting discrete logarithms, based on "double decker" exponentiation. This method is no more efficient than the one we propose; moreover, it is inadequate for our purposes, for the following reasons:

- The scheme only works for discrete logarithms, and requires all players in the system to work in the same group, which makes the scheme unacceptable in a heterogeneous environment.
- The scheme only provides a weak form of semantic security: Stadler only proves (under reasonable intractability assumptions) that it is hard to compute a discrete logarithm from its encryption. This *does not* imply that the scheme may be safely used as an *escrow* scheme.

A scheme quite similar to Stadler's is presented by Young and Yung [YY98]. Bao, Deng, and Mao [BDM98] use Stadler's scheme in fair exchange protocol. All of the papers [Sta96, YY98, BDM98] inappropriately recommend a verifiable *encryption* scheme like Stadler's for the purpose of verifiable *escrow*, which is simply not justified. These papers erroneously overlook the need for security against chosen ciphertext attack.

## 2.2 A simple but flawed fair exchange protocol

Given a method for verifiably escrowing a digital signature, the first thing one might think of is a protocol that runs something like this. Let $A$ and $B$ be the two players holding signatures they wish to exchange. We assume they have already agreed on the messages to be signed, and the public keys to be used, and we let $A$ go first.

1. $A$ sends $B$ a verifiable escrow of his digital signature, attaching a condition that describes the signature that $B$ is supposed to give $A$.
2. $B$ receives and verifies the escrow from $A$. If this succeeds, $B$ sends $A$ his signature. Otherwise, $B$ quits.

3. $A$ receives and verifies the signature from $B$. If this succeeds, $A$ sends $B$ his signature, outputs the signature from $B$, and quits.

4. $B$ receives and verifies the signature from $A$. If this succeeds, $B$ outputs the signature from $A$ and quits. Otherwise, $B$ takes the escrow it received in step (2) to the trusted third party for decryption, and $B$ deposits his signature with the trusted third party so that $A$ can retrieve it at a later time (the condition that $A$ attached to the verifiable escrow ensures that $B$ deposits the correct signature).

The logic of this protocol is essentially the same as the that of the protocols in [ASW97, BDM98, BP90, Mic97]. Actually, the protocols in [ASW97, BP90, Mic97] are in a somewhat different setting, and do not make use of the notion of verifiable escrow in exactly the same way as we do, but that does not affect the point we wish to make here. Although this protocol ensures fairness, it suffers from a severe—if somewhat subtle—problem. Namely, the protocol does not ensure a timely termination. To see this, consider what happens when $A$ sends his verifiable escrow in step (1), and is waiting for $B$'s signature in step (3). The question is: how long should $A$ wait? On the one hand, $B$ may have crashed or $A$'s message to $B$ may have been irretrievably lost, and $B$ might never respond. On the other hand, $B$ may have received $A$'s message, and chosen not to respond, but rather, to wait for some amount of time, after which it will go to the trusted third party to have the verifiable escrow decrypted. Thus, $A$ must in principle wait *forever* in step (3), from time to time polling the trusted third party to check if $B$ has deposited the signature (alternatively, the trusted third party can notify $A$ when the signature is deposited).

This is clearly an unsatisfactory situation. In light of this, the second thing one might think of is to incorporate some kind of *time limit*. In this "solution," the condition that $A$ attaches to the verifiable escrow would indicate a time limit—after this time limit, the trusted third party would refuse to decrypt it. Besides the usual problem of maintaining synchronized clocks, there is yet another problem. The question is: how long should this time limit be? If it is too short, then $B$ is put at risk: if $B$ needs to go to the trusted third party, but the network is temporarily down, the time limit may expire before $B$ can reach the trusted third party, and the protocol no longer ensures fairness for $B$. So we must choose a time limit large enough to accommodate such temporary, but extended network failures. If the network is the Internet, then surely a time limit on the order of *days* is necessary to ensure fairness to an acceptable degree. However long the chosen time limit, $A$ may be forced to wait in step (3) for this length of time, just as it did above (regardless of the status of the network). Thus, $B$ wants the time limit to be as large as possible, to ensure fairness, and $A$ wants the time limit to be as short as possible, to ensure timely termination. These contradictory goals can be reasonably achieved only if the communications network is extremely reliable and provides very strong latency guarantees, otherwise—as in the case of the Internet—they can not.

### 2.3 Our new protocol

Our new protocol for fair exchange ensures timely termination, without making any assumptions about the network, other than the assumption that a player can *eventually* reach the trusted third party. Indeed, each player can unilaterally terminate the protocol at any point in time, either by simply terminating, or by contacting the third party and then terminating.

We now sketch the high-level logic of our protocol.

1. $A$ sends $B$ an ordinary (non-verifiable) escrow of its signature. The condition that $A$ attaches to this escrow describes the signature that $B$ is supposed to give $A$.

2. $B$ receives the escrow from $A$. If this succeeds, $B$ sends $A$ a verifiable escrow of its signature, attaching a condition that includes the escrow that $B$ received from $A$, along with a description of what is *supposed* to be inside that escrow; otherwise, if $B$ does not receive the escrow from $A$, it quits.

3. $A$ receives and verifies the escrow from $B$. If this succeeds, $A$ sends $B$ its signature. Otherwise, $A$ contacts the trusted third party, requesting that it *abort* this transaction (see below), and then quits.

4. $B$ receives and verifies the signature from $A$. If this succeeds, $B$ sends $A$ its signature, outputs the signature from $A$, and quits. Otherwise, $B$ contacts the trusted third party, requesting that it *resolve* this transaction (see below), and then quits.

5. $A$ receives and verifies the signature from $B$. If this succeeds, $A$ outputs the signature from $B$ and quits. Otherwise, $A$ contacts the trusted third party, requesting that it *resolve* this transaction (see below), and then quits.

This protocol makes use of three sub-protocols: an *abort* protocol for $A$, a *resolve* protocol for $B$, and a *resolve* protocol for $A$.

The *abort* protocol for $A$ is a request that the trusted third party block any *future* request by $B$ to resolve the transaction. However, if $B$ has *already* resolved the transaction, then $A$ will get the signature it wants, since it is deposited by $B$ with the trusted third party during $B$'s resolve protocol. To ensure fairness, an abort blocks any attempt by $A$ to resolve, and vice versa. Also, the implementation must ensure that *only* $A$ can request an abort.

In the *resolve* protocol for $B$, $B$ requests that the trusted third party decrypt the escrow that $B$ received in step (2). In exchange for this, $B$ must deposit the signature described in the condition of this escrow. Of course, since this is an ordinary escrow, and not a verifiable escrow, there is no guarantee that what is inside this escrow is correct. In that case, $B$'s attempt to resolve simply fails, but any attempt by $A$ to resolve will also fail. If $A$ has performed an abort operation before $B$ attempts this resolve operation, then $B$'s attempt to resolve also fails.

In the *resolve* protocol for $A$, $A$ requests that the trusted third party decrypt the verifiable escrow that $A$ received in step (3). Before doing this, the trusted third party checks that the escrow that $A$ sent to $B$ in step (1) was correct; it does this using the condition attached to the verifiable escrow. Also, the implementation must ensure that only $A$ can do this resolve.

So we see that the problem of timely termination can be effectively solved at little additional expense—just one extra flow in the protocol, consisting of an *ordinary* escrow, as opposed to a more expensive verifiable escrow.[1] The protocol could be criticized because it requires the third party to keep state, but that criticism already applies to the protocol in §2.2 (although in [Mic97] it is mistakenly claimed otherwise). Indeed, if $B$ deposits its signature with the third party in step (4) of that protocol, then the third party must retain this signature until it is retrieved by $A$ (who in the meantime may have temporarily disappeared from the network). In theory, for both protocols, the information associated with any one transaction must be maintained by the third party forever. In practice, for both protocols, the information associated with a transaction can be safely flushed after some relatively long amount of time (or simply stored on tape, say).

[1] This improves on our original scheme in the extended abstract [ASW98], which required a total of two verifiable escrows.

We shall also see that our protocol can be easily adapted so as to allow the exchange of *any* secret satisfying a particular, efficiently computable predicate, in exchange for a homomorphic pre-image—or any secret that can be reduced to a homomorphic pre-image. This allows us to exchange digital content, and also yields a certified e-mail protocol.

### 2.4 Formal models and proof techniques

We address a number of rather subtle issues that have been overlooked or inadequately addressed by other works on fair exchange and related problems. In order to adequately address these issues, we define a formal model of secure fair exchange, and rigorously analyze our protocols in this model.

Previous papers on the fair exchange problem do not even formally define what fair exchange means in a truly rigorous way, let alone offer any mathematical proof of security. One approach to doing this would be to use the formal modeling techniques typified by BAN logic [BAN90]; however, this is not the approach we adopt, as we feel that this approach, while being useful in finding weaknesses in protocols, can never provide a meaningful guarantee of security.

Instead, our approach is that of modern theoretical cryptography, based on complexity theory. In this approach, one explicitly states the assumptions made about the communications network, and the power of the "attacker" or "adversary"—its computational power as well how it may interact with the system. Additionally, one explicitly states the security goal, i.e., what it means to "break" the system.

We define *security* in terms of *fairness* and *completeness*.

To define fairness, we let an adversary play the role of a corrupt player, and give it complete control over the network, arbitrarily interacting with the trusted third party, and arbitrarily delaying the honest party's requests to the trusted third party. Intuitively, fairness means that it is infeasible for the adversary to get the honest player's signature, without the honest player getting the adversary's signature.

Completeness means that if neither player is corrupt, and no messages are lost, then the exchange is successful.

To analyze our protocols, we make explicit intractability assumptions about the difficulty of breaking certain low-level cryptographic transformations, and with these assumptions, we prove that the adversary cannot break the system.

As we want our protocols to be as practical as possible, and still be able to give a convincing argument that they cannot be broken, we will in fact analyze the security of our protocols using the so-called *random hash function* model. In this model, we treat a cryptographic hash function *as if* it were a "black box" that contains a random function. This model was used by Fiat and Shamir [FS87], who showed that one could effectively collapse an interactive challenge/response protocol into a non-interactive protocol in this model. This technique is often called the "Fiat-Shamir heuristic." The random hash function model was later more fully developed and formalized by Bellare and Rogaway [BR93], who use the term *random oracle* model. A wide variety of cryptographic primitives and protocols have subsequently been analyzed in this model.

A proof of security in the random hash function model is only a heuristic proof—it may be possible to break the system without invalidating the underlying intractability assumptions. However, we believe that it still provides strong evidence of security, and is certainly better than no proof at all.

As we shall see, the random hash function model is not essential—wherever we make use of it, we shall also briefly indicate how it can be avoided at a reasonable cost.

### 2.5 Outline of the rest of the paper

The rest of this paper is organized as follows.

In §3, we present our formal model for secure fair exchange of digital signatures.

In §4, we describe the formal security requirements of any scheme that reduces the "promise" of a signature to the "promise" of a homomorphic pre-image, and present and analyze reduction schemes for a number of common signature schemes.

In §5, we recall the definition of security against adaptive chosen ciphertext attack for a public key encryption scheme, and formally define the distinct, but very closely related notion of an *escrow* scheme, and show how to build an escrow scheme out of an encryption scheme.

In §6, we define the formal security requirements of any scheme that verifiably escrows a homomorphic pre-image, and present and analyze a scheme satisfying these requirements.

In §7, we present and analyze our new protocol for fair exchange, using the definitions and constructions from the previous sections.

In §8, we discuss a variation of the fair exchange problem, where digital data, rather than a digital signature, is to be exchanged. Two specific variants are treated: digital content and certified e-mail.

In §9 we discuss an alternative method of verifiable escrow that is more efficient than the one in §6, but requires more interaction with the trusted third party.

In §10, we present a protocol for contract signing that is more efficient than our general protocol, and also has additional security features.

We conclude with some directions of future research in §11.

## 3 A Formal Security Model for Fair Signature Exchange

We have two players $A$ and $B$, and a trusted third party $T$ that acts as a server: it receives a request from a client, updates its internal state, and sends a response back to the client. $T$ has a public key known to $A$ and $B$, and is always assumed to be honest and (eventually) available. We assume the client/server channel is private, and that server responses are authenticated, but *do not* assume that the client requests are authenticated.

The two players agree upon the signatures they want to exchange, and then exchange messages back and forth. Between the time that a player receives a message and generates its response, it may send requests to $T$, obtaining the corresponding responses within a finite, but unbounded, amount of time.

We stress that we make no assumptions about a "public key infrastructure," other than the assumption that both $A$ and $B$ know $T$'s public key. Players may make use of a public key infrastructure to negotiate the signatures they wish to exchange, but that is not an issue that affects the fair exchange protocol itself.

We define *security* in terms of *fairness* and *completeness*.

As already mentioned in §2.4, fairness intuitively means that it is infeasible for a dishonest player to get the honest player's signature, without the honest player getting the dishonest player's signature, and completeness intuitively means that if neither player is corrupt, and no messages are lost, then the exchange is successful.

We now make the above notions more precise.

**Behavior of $T$.** $T$ is a polynomial-time interactive Turing machine that follows the program prescribed for it by the protocol. $T$ acts as a server, repeatedly accepting a request, updating its internal state, and generating a response. We assume that each request is processed *atomically*.

$T$ has a public key/private key pair $(PK_T, SK_T)$ that is generated by a key generation algorithm prescribed by the protocol.

**Behavior of an honest player.** An honest player is a polynomial-time interactive Turing machine that follows the program prescribed for it by the protocol. It interacts with its environment through a sequence of rounds: in one round it receives a message, updates its internal state, and generates a response.

Before generating a response, it may access $T$ (perhaps several times). To do this, the player must explicitly signal its intention to contact $T$, and then wait on an externally generated signal before proceeding.

The initial state of an honest player is determined by a set of inputs: $PK_T, PK, m, \sigma, PK', m'$. Here, $\sigma$ is a signature on message $m$ under the public key $PK$ that the player intends to give in exchange for a signature on $m'$ under the public key $PK'$.

After a bounded number of rounds, an honest player stops and writes on a private tape an output $\sigma'$. The player also externally signals that it has terminated.

**Definition of fairness.** Fix a particular signature scheme $\Sigma$, and consider the following game. The players in the game are an adversary, called $B^*$, which is a polynomial-time interactive Turing machine, an honest player, called $A$, as well as $T$, $T$'s key generation algorithm, $\Sigma$'s key generation algorithm, and a corresponding signing oracle $S$. Note that all the players other than $B^*$ are purely *reactive*, i.e, they respond to events that are triggered by $B^*$; thus, it is $B^*$ that drives the game forward.

### Game A

A1 Run $\Sigma$'s key generation algorithm, giving the secret key to a signing oracle $S$ and the public key $PK$ to $B^*$. Also generate $T$'s public and private keys, giving $SK_T$ to $T$ and $PK_T$ to $B^*$.

A2 $B^*$ interacts arbitrarily with $T$ and $S$, obtaining signatures on adaptively chosen messages.

A3 $B^*$ selects messages $m$ and $m'$, and an arbitrary public key $PK'$ for a signature scheme (possibly different from $\Sigma$). The message $m$ must be different from those given to $S$ in A2. Now, $S$ produces a signature $\sigma$ on $m$, and $A$ is initialized with inputs: $PK_T, PK, m, \sigma, PK', m'$. The signature $\sigma$ is not seen by $B^*$.

A4 $B^*$ interacts with $T$, $S$ and $A$ in an arbitrary fashion, subject to the following restrictions:

(1) $B^*$ may not query $S$ with $m$.

(2) When $A$ signals its intention to contact $T$, $B^*$ must eventually signal $A$ to let it proceed, after which $B^*$ refrains from contacting $T$ until $A$ is finished (i.e., generates a response or a signal).

(3) Unless $A$ has signaled termination, $B^*$ must eventually supply another input message.

Eventually, $A$ terminates and outputs a string $\sigma'$, and $B^*$ also terminates, and outputs a string $\bar{\sigma}$. We say that $B^*$ wins the game if $\sigma'$ is not a valid signature for $m'$ under $PK'$, but $\bar{\sigma}$ is a valid signature for $m$ under $PK$. We define *fairness* to mean that the probability that $B^*$ wins this game is negligible (with respect to some security parameter).

**Remark 1** Restriction (2) is the rather technical embodiment of our intuitive requirement that $A$ can always reach $T$, but may be arbitrarily delayed in doing so.

**Remark 2** Our formal model contains no notion of absolute time; however, any real-world protocol should be able to "time out" if it has waited an excessively long amount of time for a message. Such a "time out" can be easily represented in our model by having $B^*$ deliver a special "time out message" to $A$. Thus, we do not view our protocols as making non-deterministic decisions: a protocol responds in a well-defined way to any given input; a "time out" is just a specific input that triggers a specific (protocol dependent) response. Another way to view this is that we separate the "high level" protocol—which responds to a "time out message"—from the "low level" communications subsystem that might actually use a clock to generate the "time out message"—and we place the latter under the control of the adversary.

**Remark 3** In our definition, the player in the exchange protocol is not necessarily the holder of the signing key. This may be useful in situations where the signature is generated by another party.

**Remark 4** In our definition, we allow $B^*$ to obtain signatures from the signing oracle on any message except $m$ itself. The reason for this is that it allows one to prove that if $A$ engages in a number of fair exchange protocols (possibly interleaved), then all of these protocol instances terminate fairly. The argument is standard, and runs as follows. If one protocol instance does not terminate fairly, we could just guess which protocol instance this is; for all other protocol instances, we use the signature oracle and run the fair exchange protocol with that signature; if the guessed protocol instance does not terminate fairly, this contradicts our definition of security.

**Remark 5** In the previous remark, we assumed that all of the signatures that $A$ wishes to use in the various protocol instances are distinct. However, if $A$ runs the protocol once, and it ends unsuccessfully, then perhaps $A$ should be allowed to run the protocol again with the same signature (to exchange with a possibly different signature than in the first run). To keep things from getting too complicated, we have not formalized this notion of security in this somewhat richer setting, although this would not be too difficult to do. Although our formal notion of security does not logically imply security in this somewhat richer setting, all of the protocols we discuss here are in fact secure in this richer setting.

**Remark 6** Although we believe our definition of secure fair exchange is natural and useful, one could attempt to define an even stronger notion of fair exchange along the lines of *simulatability* (see, e.g., [Bea91]). In this approach, one would define an *ideal* fair exchange protocol that uses an on-line trusted third party. A *real* protocol would then be proven secure by showing that any attack on the real protocol could be effectively transformed into an equivalent attack on the ideal protocol. It seems to be reasonably challenging to work out all the details of such a definition, and we have not done so; however, such a definition (whatever the details) seems to be substantively stronger than our definition, and it is not at all clear that our protocols achieve this level of security.

**Remark 7** Of course, the names $A$ and $B^*$ are arbitrary, and sometimes it will be convenient to use the names $B$ and $A^*$ instead.

**Definition of completeness.** We define another game similar to that above, in which the adversary gets access to two signing oracles, $S$ and $S'$, and initializes two honest players $A$ and $B$, who interact directly with each other. The adversary in this case can interact with $T$, but cannot interfere with the interaction of $A$ and $B$, except insofar as the adversary still has the power to schedule both $A$'s and $B$'s interactions with $T$. We omit the details of this game, which are straightforward. The real-world situation that this game models is that where all messages are delivered without being seen or modified by the adversary, and neither player "times out" waiting for a message (see Remark 2).

We define *completeness* to mean that it is infeasible for the adversary in the above game to prevent $A$ and $B$ from successfully exchanging their signatures. We further say that a fair exchange protocol is *optimistic* if in this game neither $A$ nor $B$ interact with $T$.

# 4 Reducing Signatures to Homomorphic Pre-images

In this section, we show how to reduce a "promise" of a signature to the "promise" of a particular homomorphic pre-image.

As a simple, motivating example, consider the standard hash-and-invert RSA signature (which is provably secure in the random hash function model [BR93]). Here, the signature on a message is simply the $e$th root of the hash of the message, viewed as a number modulo a given composite number. In this case, reduction is trivial: the promised homomorphic pre-image is simply this $e$th root.

Reduction schemes for other signature schemes are not so trivial. To be able to uniformly treat several signature schemes, we start with a formal definition of our requirements.

## 4.1 Definition of a secure reduction

Let $\Sigma$ be a signature scheme. A *reduction scheme* for $\Sigma$ consists of three efficient algorithms, a *reduction algorithm* $\mathsf{reduce}$, a *verification algorithm* $\mathsf{verify}$, and a *recovery algorithm* $\mathsf{recover}$.

- $\mathsf{reduce}$ takes as input $PK, m, \sigma$, where $PK$ is a public key for $\Sigma$, $m$ is a message, and $\sigma$ is signature on $m$ under $PK$. The output consists of a surjective homomorphism $\theta : G_1 \to G_2$ of commutative groups, $d \in G_2$, $c \in \{0,1\}^*$, and $s \in \theta^{-1}(d)$. The string $c$ encodes additional information that is used by the verification and recovery algorithms. The group homomorphism $\theta$ is described in some canonical way (including descriptions of $G_1$ and $G_2$), and should be efficiently computable.
- $\mathsf{verify}$ takes as input $PK, m, \theta, d, c$, and either accepts or rejects.
- $\mathsf{recover}$ takes as input $PK, m, \theta, c$, and $s \in G_1$, and outputs a string $\bar{\sigma}$.

A *secure* reduction scheme should satisfy three properties:

**Completeness.** If $\mathsf{reduce}(PK, m, \sigma) = (\theta, d, c, s)$, then $\mathsf{verify}(PK, m, \theta, d, c)$ accepts.

**Soundness.** It is infeasible for an adversary to find $PK, m, \theta, d, c, s$ such that $\mathsf{verify}(PK, m, \theta, d, c)$ accepts and $\theta(s) = d$, but $\mathsf{recover}(PK, m, \theta, c, s)$ is not a valid signature on $m$.

**Secrecy.** It is infeasible for an adversary to win the following game:

### Game B

B1 Run $\Sigma$'s key generation algorithm, giving the secret key to a signing oracle $S$ and the public key $PK$ to the adversary.

B2 The adversary makes arbitrary queries to $S$.

B3 The adversary generates a message $m$ different from those given to $S$ in B2. Now $S$ generates a signature $\sigma$ on $m$ under $PK$, and the adversary is given $d, c$, where $(\theta, d, c, s) = \mathsf{reduce}(PK, m, \sigma)$.

B4 The adversary continues to query $S$ on messages different from $m$.

The adversary wins the game if it can output a valid signature on $m$.

**Remark 8** Clearly, our definition of a secure reduction scheme implies that the underlying signature scheme is secure against adaptive chosen-message attacks [GMR88]. A stronger definition could be formulated wherein the reduction scheme is "just as secure" as the underlying signature scheme, however secure it happens to be. Unfortunately, not all of our proofs achieve this. It is also possible to consider more general reduction schemes; for example, the reduction procedure could be interactive. We do not consider these here. Note that for some reduction schemes, the group homomorphism $\theta$ depends only on the public key, while for others, it may vary with the signature (see 4.3).

### 4.2 Schnorr Signatures

In the Schnorr signature scheme, to generate a public key, one selects primes $p$ and $q$ such that $q \mid p-1$, and a generator $g$ for the subgroup of $\mathbf{Z}_p^*$ of order $q$. One then chooses $x \in \mathbf{Z}_q$ at random and computes $h = g^x$. The public key consists of $p, q, g, h$, and the private key is $x$. To sign a message $m$, the signer chooses $r \in \mathbf{Z}_q$ at random, and computes $z = cx + r$, where $c = H(g^r, m) \in \mathbf{Z}_q$ and $H$ is a hash function. The signature is $(c, z)$. To verify a signature, one checks that $c = H(g^z h^{-c}, m)$.

The Schnorr signature scheme is provably secure in the random hash function model (if the discrete logarithm problem is hard).

The following reduction from Schnorr signatures to discrete logarithms was observed in [FR95], in the context of verifiable signature sharing. The reduction algorithm takes as input a signature $(c, z)$ on a message $m$, and outputs $u = g^z, c$, where the promised pre-image is $z = \log_g u$.

The verification algorithm checks that $c = H(uh^{-c}, m)$. Given $z$, the recovery algorithm outputs $(c, z)$.

Completeness and soundness are clear. Secrecy follows from the fact that one can simulate the "public" output of the reduction algorithm without a signature: the simulator chooses $r \in \mathbf{Z}_q$ at random, computes $w = g^r$ and $c = H(w, m)$, and sets $u = wh^c$.

### 4.3 DSS Signatures

Key generation for DSS is identical to that for the Schnorr scheme. The standard prescribes that $q$ has a length of 160 bits. To sign a message $m$, the signer chooses $k \in \mathbf{Z}_q^*$ at random, and computes $r = (g^k) \bmod q$ and $s = k^{-1}(H(m) + xr)$, where $H$ is a hash function with outputs in $\mathbf{Z}_q$. The signature is $(r, s)$. A signature is verified by checking that $r = (g^{u_1} h^{u_2}) \bmod q$, where $u_1 = H(m)s^{-1}$, and $u_2 = rs^{-1}$. Note that signatures with $s = 0$ are invalid, but these effectively never arise.

We now give a reduction scheme reducing a promise of a DSS signature to a promise of discrete logarithm.

We have a signature $(r, s)$ on a message $m$. Let $u_1, u_2$ be defined as above. The reduction algorithm outputs $\alpha = g^{u_1} h^{u_2} \in \mathbf{Z}_p^*$. The promised homomorphic pre-image is $s = \log_\alpha \beta$ where $\beta = g^{H(m)} h^r \in \mathbf{Z}_p^*$.

The verification algorithm, given $\alpha$, computes $r = \alpha \bmod q$ and $\beta = g^{H(m)} h^r$, and checks that $\alpha^q = 1$, $\alpha \neq 1$ (so $\alpha = g^k$ for $k \in \mathbf{Z}_q^*$), and $\beta \neq 1$ (so $s \neq 0$).

Given the discrete logarithm $s = \log_\alpha \beta$, then it is easy to see that $(r, s)$ is a valid signature on $m$.

Completeness and soundness are clear. Secrecy follows from the fact that the " public" output of the reduction algorithm is easily simulatable: the simulator chooses $k \in \mathbf{Z}_q^*$ at random and outputs $\alpha = g^k$.

While this reduction scheme is adequate, one property of this scheme is that the base of the relevant discrete logarithm, namely $\alpha$, is not fixed. As we shall see in §9, it is sometimes convenient to have a fixed base (and hence a fixed group homomorphism). This can be easily achieved by making a change of base: the reduction protocol outputs $\alpha$, as well as $\gamma = g^s$, and appends a proof that $\log_\alpha \beta = \log_g \gamma$. The promised discrete logarithm is then $s = \log_g \gamma$.

For the proof of equality of discrete logarithms, we can use a well-known interactive honest-verifier zero-knowledge proof system (see [CP92]), and applying the Fiat-Shamir heuristic, this becomes a non-interactive zero-knowledge proof in the random hash function model.

To prove secrecy, we show how to simulate the output of the reduction scheme as follows. We compute $\alpha = g^k$ for random $k \in \mathbf{Z}_q^*$, $r = \alpha \bmod q$, $\beta = g^{H(m)} h^r \in \mathbf{Z}_p^*$, and $\gamma = \beta^{k'}$, where $k' = k^{-1} \bmod q$. The value $s$ (which the simulator does not know) is determined by the relation $\alpha^s = \beta$, and since $\alpha = g^k$ (where the simulator *does* know $k$), we have

$$\gamma = \beta^{k'} = g^{ksk'} = g^s.$$

Thus, the values $\alpha$, $\beta$, and $\gamma$ have the correct distribution; in particular, $\log_\alpha \beta = \log_g \gamma$. Finally, we append to the output a simulated proof of equality of discrete logarithms.

### 4.4 Fiat-Shamir Signatures

In the Fiat-Shamir signature scheme, there are two security parameters $k$ and $t$, and a hash function $H$ that outputs a $t \times k$ bit-matrix. To generate a key, one chooses a composite modulus $M$, and for $1 \le j \le k$, selects $s_j \in \mathbf{Z}_M^*$ at random, and computes $v_j = s_j^2$. The public key consists of $M$ and $v_1, \ldots, v_k$, and the secret key consists of $s_1, \ldots, s_k$. To sign a message $m$, for $1 \le i \le t$ the signer chooses $r_i \in \mathbf{Z}_M^*$ at random, computes $w_i = r_i^2$, and then computes $c = H(m, w_1, \ldots, w_t)$. Next, for $i = 1, \ldots, t$, the signer computes $z_i = r_i / \prod_{j=1}^k s_j^{c_{ij}}$. The signature is $(c, z_1, \ldots, z_t)$. To verify a signature, one checks that $H(m, w_1, \ldots, w_t) = c$, where $w_i = z_i^2 \prod_{j=1}^k v_j^{c_{ij}}$.

The Fiat-Shamir scheme is provably secure in the random hash function model (if factoring is hard).

Our reduction algorithm works as follows. We have a signature $(c, z_1, \ldots, z_t)$ on a message $m$. We output

$$u_1 = z_1^2;\ u_2 = z_1 z_2, \ldots, u_t = z_1 z_t,\ c.$$

The promised pre-image is a square root of $u_1$.

The verification algorithm checks that

$$H(m, u_1 \prod_{j=1}^k v_j^{c_{1j}}, u_2^2/u_1 \prod_{j=1}^k v_j^{c_{2j}}, \ldots, u_t^2/u_1 \prod_{j=1}^k v_j^{c_{tj}}) = c.$$

The recovery algorithm takes a square root $z_1$ of $u_1$, and computes $z_i = u_i/z_1$ for $2 \le i \le t$, and outputs the signature $(c, z_1, \ldots, z_y)$.

To prove that this scheme achieves secrecy, we have to make the following *strong security assumption* about the Fiat-Shamir scheme: given signatures on several messages, not only is it difficult to compute a signature on a new message, it is difficult to compute a signature on any of the given messages with a different $(w_1, \ldots, w_t)$. This strong security assumption holds in the random hash function model if factoring is hard.

**Lemma 1** *Under the strong security assumption for the Fiat-Shamir scheme, the above reduction scheme is secure.*

Completeness and soundness are trivial.

To prove secrecy, suppose that an adversary can win Game B. Let $m$ be the message chosen in B3 by the adversary, and let $u_1, \cdots, u_t, c$ be the output of the reduction algorithm. Under the strong security assumption for Fiat-Shamir signatures, the only signature that the adversary can compute is one of the form $(c, z_1', \ldots, z_t')$, where $z_1'$ is a square root of $u_1$. We show how to use this adversary to factor $M$.

We start playing Game B against the adversary by generating a public key/secret key pair for the signature scheme. Since we know the secret key, we can generate signatures on demand. It is straightforward to verify that the reduction algorithm is *witness hiding*, in the sense that the $z_1$ used by the reduction algorithm is distributed uniformly among the square roots of $u_1$, independently of the output of the reduction algorithm (a similar independence condition holds for the other $z_i$, but *not* for their joint distribution). Therefore, $z_1$ and $z_1'$ are independent square roots of $u_1$, and so with probability 1/2, $\gcd(M, z_1 - z_1')$ is a proper factor of $M$.

That completes the proof of Lemma 1.

### 4.5 Ong-Schnorr Signatures

Ong-Schnorr signatures are a generalization of Fiat-Shamir signatures with a third security parameter $l$. The hash function $H$ outputs a $t \times k$ matrix, with each entry in $\{0, \ldots, 2^l - 1\}$. To generate a key, one chooses a composite modulus $M$, and for $1 \le j \le k$, selects $s_j \in \mathbf{Z}_M^*$ at random, and computes $v_j = s_j^{2^l}$. The public key consists of $M$ and $v_1, \ldots, v_k$, and the secret key consists of $s_1, \ldots, s_k$. To sign a message $m$, for $1 \le i \le t$ the signer chooses $r_i \in \mathbf{Z}_M^*$ at random, computes $w_i = r_i^{2^l}$, and then computes $c = H(m, w_1, \ldots, w_t)$. Next, for $i = 1, \ldots, t$, the signer computes $z_i = r_i / \prod_{j=1}^k s_j^{c_{ij}}$. The signature is $(c, z_1, \ldots, z_t)$. To verify a signature, one checks that $H(m, w_1, \ldots, w_t) = c$, where $w_i = z_i^{2^l} \prod_{j=1}^k v_j^{c_{ij}}$.

The Ong-Schnorr signature scheme is provably secure in the random hash function model (if factoring is hard). This was proven for so-called "Blum integers" (where $M$ is a product of two primes equal to 1 modulo 4) by Jakobsson [Jak92], and later extended to more general moduli by Schnorr [Sch96, Sch97]; the corresponding interactive identification scheme is proved secure in [Sho96].

Our reduction algorithm works as follows. We have a signature $(c, z_1, \ldots, z_t)$ on a message $m$. We output

$$u_1 = z_1^2;\ u_2 = z_1 z_2, \ldots, u_t = z_1 z_t,\ c.$$

The promised pre-image is a square root of $u_1$.

The verification algorithm checks that

$$H(m, u_1^{2^{l-1}} \prod_{j=1}^{k} v_j^{c_{1j}}, (u_2^2/u_1)^{2^{l-1}} \prod_{j=1}^{k} v_j^{c_{2j}}, \ldots, (u_t^2/u_1)^{2^{l-1}} \prod_{j=1}^{k} v_j^{c_{tj}}) = c.$$

The recovery algorithm takes a square root $z_1$ of $u_1$, and computes $z_i = u_i/z_1$ for $2 \le i \le t$, and outputs the signature $(c, z_1, \ldots, z_y)$.

We have to make the same strong security assumption that we made for the Fiat-Shamir scheme (which also is provable in the random hash function model).

**Lemma 2** *Under the strong security assumption for the Ong-Schnorr scheme, the above reduction scheme is secure.*

The proof is essentially the same as that for Lemma 1, and we omit the details.

### 4.6 GQ Signatures

GQ signatures are essentially isomorphic to Schnorr signatures, with the $e$th-power map modulo a composite in GQ playing the role of exponentiation in Schnorr. Here, $e$ should be a large prime number. Because of this isomorphism, the reduction from GQ signatures to $e$th roots is isomorphic to the reduction from Schnorr signatures to discrete logarithms. We leave the details to the reader.

### 4.7 Anonymous Off-Line Electronic Coins

Various schemes have been proposed for anonymous, off-line cash. For concreteness, we consider a scheme of Brands [Bra93]. In this scheme, there is a group $G$ of order $q$, and two generators $g_1, g_2 \in G$. A coin consists of two group elements $A, B \in G$, along with a special kind of signature on the coin from the bank. To spend a coin at a given shop, a user generates a payment transcript consisting of $r_1, r_2 \in \mathbf{Z}_q$ such that $g_1^{r_1} g_2^{r_2} = A^d B$, where $d$ is a hash of the coin, the shop's identity, the current data and time, and possibly other details of the transaction.

Let $\theta : \mathbf{Z}_q \times \mathbf{Z}_q \to G$ be the group homomorphism sending $(x_1, x_2)$ to $g_1^{x_1} g_2^{x_2}$. Then reducing a promise of a payment transcript to a promise of a homomorphic pre-image is trivial: the promised pre-image is a $\theta$-pre-image of $A^d B$.

## 5 Public Key Encryption and Escrow Schemes

In this section, we recall the notion of security against chosen ciphertext attack for public key cryptosystems, and extend this notion to that of a secure *escrow* scheme.

### 5.1 Chosen ciphertext security

A *semantically secure* public key encryption scheme [GM84] ensures security against a *passive* adversary who can only eavesdrop. This notion of security is not strong enough for our application. The notion we need is that of *security against chosen ciphertext attack* [RS91, DDN91], which ensures security against an *active* adversary. We recall here the definition of this notion.

A public key cryptosystem consists of a key generation algorithm, an encryption algorithm, and a decryption algorithm.

The key generation algorithm generates a public key/private key pair $(PK, SK)$.

The encryption algorithm $E$ is a function that takes as input the public key $PK$, a random bit string $t$, and a message $m$, and produces the ciphertext $\psi = E(PK, t, m)$.

The decryption algorithm $D$ is a function that takes as input the private key $SK$ and a ciphertext $\psi$ and outputs a message $m = D(SK, \psi)$. We assume that for all $t$ and $m$, $D(SK, E(PK, t, m)) = m$. The decryption algorithm may output "error."

The attack scenario is as follows. The adversary interacts with a decryption oracle, obtaining decryptions of arbitrary ciphertexts $\psi'$ of his choosing. Note that $\psi'$ can be chosen in an arbitrary way; in particular, it need not be an output of the encryption algorithm. Then the adversary selects two messages $m_0, m_1$ (of equal length), and gives these to an encryption oracle, which chooses a random bit $b$ and returns an encryption $\psi$ of $m_b$ to the adversary. The adversary then proceeds to feed more ciphertexts $\psi'$ to the decryption oracle, subject only to the restriction that $\psi' \neq \psi$. At the end of the game the adversary attempts to guess $b$. Security means that the adversary cannot guess $b$ correctly with probability significantly greater than 1/2.

The OAEP encryption scheme of Bellare and Rogaway [BR94] is a practical scheme that is provably secure in this sense in the random hash function model (if the RSA inversion problem is hard). The scheme of Cramer and Shoup [CS98] is also practical, although slightly less efficient; however, it can be proved secure without resorting to the random hash function model (if the Decisional Diffie-Hellman problem is hard).

### 5.2 Secure escrow

An escrow scheme consists of a key generation algorithm, an encryption algorithm, and a decryption algorithm.

The key generation algorithm generates a public key/private key pair $(PK, SK)$.

The encryption algorithm $E$ is a function that takes as input the public key $PK$, a random bit string $t$, and a message $m$, and a condition $\kappa$, and produces the ciphertext $\psi = E(PK, t, m, \kappa)$. The condition $\kappa$ is an arbitrary bit string.

The decryption algorithm $D$ is a function that takes as input the private key $SK$, a ciphertext $\psi$, and a condition $\kappa$, and outputs a message $m = D(SK, \psi, \kappa)$. We assume that for all $t$, $m$, and $\kappa$, $D(SK, E(PK, t, m, \kappa), \kappa) = m$. The decryption algorithm may output "error."

The attack scenario is as follows. The adversary interacts with a decryption oracle, obtaining decryptions of arbitrary ciphertext/condition pairs $(\psi', \kappa')$ of his choosing. Then the adversary selects two messages $m_0, m_1$ (of equal length), along with a condition $\kappa$, and gives these to an encryption oracle, which chooses a random bit $b$ and returns an encryption $\psi$ of $m_b$ using the condition $\kappa$ to the adversary. The adversary then proceeds to feed more ciphertext/condition pairs $(\psi', \kappa')$ to the decryption oracle, subject only to the restriction that $(\psi', \kappa') \neq (\psi, \kappa)$. At the end of the game the adversary attempts to guess $b$. Security means that the adversary cannot guess $b$ correctly with probability significantly greater than 1/2.

Given an encryption scheme that is secure against chosen ciphertext attack, it is easy to construct a secure escrow scheme as follows. Let $PK, SK, E, D$ be the public key, secret key, encryption algorithm, and decryption algorithm of the public key encryption scheme. Let $H$ be a collision-resistant hash function. To escrow a message $m$ under condition $\kappa$, we simply compute the ciphertext $\psi = E(PK, t, (m, H(\kappa)))$. Given a ciphertext $\psi$ and condition $\kappa$, if $D(SK, \psi)$ is of the form $(m, H(\kappa))$, the decryption algorithm in the escrow scheme outputs $m$; otherwise, it outputs "error."

It is easy to show that the resulting scheme is a secure escrow scheme in the sense we have just defined.

**Remark 9** Note that secure escrow cannot be achieved by having the entity who creates the escrow sign $(\psi, \kappa)$, where $\psi$ is just a semantically secure encryption of $m$. Another entity could simply present $(\psi, \kappa')$ signed under some other signature key to the escrow decryption service—there is nothing that securely binds $\kappa$ to $\psi$. This binding is essential. It can be achieved as described above using a chosen ciphertext secure encryption scheme. It could also be achieved using an encryption scheme that is only semantically secure if one allows *pre-registration* with the escrow decryption service. This would work as follows. When a player pre-registers with the escrow service, he obtains a certificate (signed by the escrow service) that contains (1) a public encryption key, (2) a public signature verification key, and (3) an encryption of the private decryption key corresponding to (1) (encrypted so that the escrow service can decrypt it); in addition to this certificate, the player obtains the private signing key corresponding to (2). Now the player escrows a secret by encrypting it using the public encryption key in the certificate, and signing the ciphertext/condition using the given signing key. Before the escrow decryption service decrypts the ciphertext, it checks the certificate's signature and the signature on the ciphertext/condition.

**Remark 10** In the context of *threshold decryption*, where the escrow decryption service is a distributed, fault-tolerant service, a different approach is required. See Shoup and Gennaro [SG98] for a solution to this problem.

# 6 Verifiable Escrow of Homomorphic Pre-images

In this section, we give a formal definition of a secure verifiable escrow scheme, give a general, yet efficient, construction that can be proven secure in the random hash function model, and discuss two particular implementations.

## 6.1 A definition of secure verifiable escrow

Suppose we have a surjective homomorphism $\theta : G_1 \to G_2$ of commutative groups (we shall use the "+" operator to denote the group operation for a generic commutative group). We have a publicly known group element $d \in G_2$ and a secret $s \in \theta^{-1}(d)$. We want to escrow $s$ under the public key of a third party in such a way that it can be publicly verified that when decrypted, a pre-image of $d$ is obtained. However, we want to ensure that this verification procedure itself does not reveal any information that makes it easier to compute an $s \in \theta^{-1}(d)$. As with an ordinary escrow, we want to bind to the encryption a *condition* $\kappa \in \{0,1\}^*$, which will be used by the third party to determine if this decryption is authorized. This condition should also be verifiable.

We will define a secure escrow scheme in terms of *completeness*, *soundness*, and *zero knowledge*. The zero-knowledge part of our definition is a bit unusual. We define a very strong notion of zero knowledge which we call "plug and play" zero knowledge.

More formally, a verifiable escrow scheme consists of a key generation algorithm, a prover $P$, a verifier $V$, and a decryption algorithm $D$. $P$ and $V$ have common inputs $d \in G_2$ and $\kappa \in \{0,1\}^*$, along with the public encryption key. $P$ also has a private input $s \in \theta^{-1}(d)$. At the end of the protocol, $V$ either accepts and outputs a string $\alpha$, or rejects. The string $\alpha$ is a ciphertext that can be given to $D$, along with the condition $\kappa$.

A *secure* verifiable escrow scheme satisfies the following properties:

**Completeness.** If both $P$ and $V$ are honest, then for all $s$, $d$, and $\kappa$, with $\theta(s) = d$, $V$ accepts.

**Soundness.** Consider the following game played by an adversary against an honest verifier $V$. The adversary has oracle access to $D$, and interacts with $V$, where the adversary chooses $d$ and $\kappa$. We say that $V$ is "cheated" in this game if $V$ accepts and outputs $\alpha$, but $\theta(D(\alpha, \kappa)) \neq d$. Soundness means that the probability that $V$ is "cheated" is negligible.

**"Plug and play" zero knowledge.** Intuitively, this notion means that we can literally replace a "real" prover with a "fake" prover so that nobody will notice. More formally, consider the following game played against an adversary:

### Game C

C1 The key generation algorithm is run, the private key is given to $D$ and the public key is given to the adversary.

C2 The adversary makes arbitrary queries $D(\alpha', \kappa')$.

C3 The adversary generates $s, d, \kappa$ with $\theta(s) = d$, and gives $P$ the input $s, d, \kappa$, along with the public encryption key.

C4 The adversary makes arbitrary queries to $D$ and $P$, but after its first query to $P$, it may not query $D$ with the condition $\kappa$.

C5 At the end of the game, the adversary outputs 0 or 1.

A "plug and play" simulator is a machine that can be "plugged into" the above game, replacing the key generation algorithm, the decryption oracle $D$, and the prover $P$, but is only given $d$ and $\kappa$—and not $s$. In the random hash function model, the simulator also responds to the random oracle queries. *"Plug and play" zero knowledge* means there exists a "plug and play" simulator such that the adversary cannot feasibly distinguish between the real game and the simulated game, i.e., the probability that it outputs a 1 in the real game should not differ significantly from the probability that it outputs 1 in the simulated game.

**Remark 11** As already mentioned, this is a much stronger notion of zero-knowledge than more traditional notions [GMR89, GO94]. In our definition, the simulator is an oracle that must respond to queries *on-line*, and is not allowed to do anything like "rewind" the adversary, nor does it have control over the adversary's random coins. This constraint makes it easy to use such a protocol as a sub-protocol in a larger protocol, and to prove the overall security of the larger protocol, without running into any of the subtle problems often associated with protocol composition. Indeed, we will want to use such a protocol in a rather complicated setting, where there are numerous, concurrently interacting system components—some components are under control of the adversary, and some are not (see Remark 22). Despite the strength of this notion, it can still be implemented rather efficiently—the random hash function model helps, but is not essential.

**Remark 12** We have defined "plug and play" zero knowledge in the random hash function model, so that the simulator also "controls" the random oracle. Note, however, that to avoid protocol composition problems, if the verifiable escrow protocol is just one component of a larger protocol, then the random oracle used by the verifiable escrow protocol should be *independent* of the random oracles used in other components. This means that when we instantiate these random oracles using cryptographic hash functions, some care should be taken to ensure that inputs to hash functions used by different components are properly segmented so as to be distinct. This is not hard to do, but it is a point not discussed at all in other papers that use the random hash function model. The problem is analogous to the problem of using the same RSA modulus for both signing and encryption.

### 6.2 A general construction

We now give a general construction for a verifiable escrow scheme.

First, we assume that we have an ordinary escrow scheme, as defined in §5. Let $PK, SK, E, D$ be the public key, secret key, encryption algorithm, and decryption algorithm of this escrow scheme. Recall that $E$ is a function that takes as input $PK$, a random string $t$ (of length, say, $k$), a message $m$, and a condition $\kappa$, and outputs a ciphertext $\psi = E(PK, t, m, \kappa)$.

Second, we assume we have hash functions $H_1$ and $H_2$. $H_1$ takes a string $r$, of length, say $l$, and outputs a pair $(t, s') \in \{0,1\}^k \times G_1$. $H_2$ hashes long strings to short strings. Also, a security parameter $N$ is defined.

Our verifiable escrow protocol is a simple "cut and choose" scheme (but with exponential—not linear—security in $N$), and runs as follows.

**Protocol D**

D1 For For $1 \leq i \leq N$, $P$ chooses $r_i \in \{0,1\}^l$ at random, and computes

- $(t_i, s'_i) = H_1(r_i)$;
- $\psi_i = E(PK, t_i, s'_i, \kappa)$;
- $h_i = (\psi_i, \theta(s'_i))$.

$P$ sends $h = H_2(h_1, \ldots, h_N)$ to $V$.

D2 $V$ receives $h$ from $P$, chooses random bits $b_1, \ldots, b_N$, not all zero, and sends $b_1, \ldots, b_N$ to $P$.

D3 $P$ receives $b_1, \ldots, b_N$ from $V$. For $1 \leq i \leq N$, $P$ computes $v_i$ as follows.
If $b_i = 0$, $v_i = r_i$.
If $b_i = 1$, $v_i = (\psi_i, s''_i)$, where $s''_i = s'_i + s$.
$P$ then sends $v_1, \ldots, v_N$ to $V$. to $V$.

D4 $V$ receives $v_1, \ldots, v_N$ from $P$. For $1 \leq i \leq N$, $V$ computes $h_i$ as follows.
If $b_i = 0$, then $h_i = (\psi_i, \theta(s'_i))$, where $\psi_i = E(PK, t_i, s'_i, \kappa)$, $(t_i, s'_i) = H_1(r_i)$, and $r_i = v_i$.
If $b_i = 1$, then $h_i = (\psi_i, \theta(s''_i) - d)$, where $(\psi_i, s''_i) = v_i$.
Then $V$ computes $\tilde{h} = H_2(h_1, \ldots, h_N)$. If $\tilde{h} \neq h$, then $V$ rejects; otherwise, $V$ accepts and outputs
$$\alpha = (d, \{(\psi_i, s''_i) : 1 \leq i \leq N,\ b_i = 1\}).$$

Decryption is straightforward. Given $\alpha$ as above, and $\kappa$, each $\psi_i$ is decrypted in turn, using the decryption algorithm $D$, the secret key $SK$, and the condition $\kappa$. If the decryption yields a group element $s'_i$ such that $\theta(s''_i - s'_i) = d$, then the decryption algorithm stops and outputs $s = s''_i - s'_i$. If all of these decryptions fail to yield a homomorphic pre-image of $d$, then the decryption algorithm outputs "error."

**Lemma 3** *In the random hash function model for $H_1$ and $H_2$, the above scheme is a secure verifiable encryption scheme.*

*Completeness.* Clear.

*Soundness.* This is a standard "cut and choose" argument, relying only on the assumption that $H_2$ is collision resistant. The probability that an honest player is "cheated" is roughly $2^{-N}$.

*Zero Knowledge.* The simulator will play the role of the prover and the random oracle $H_2$. We only require that $H_1$ act as a pseudo-random generator.

As $H_2$ is a random oracle, the simulator is free to send the adversary a random string $h$ in step D1, and then wait for the adversary to send back $b_1, \ldots, b_N$—the simulator, unlike a real-world prover, is not forced to commit to anything at all. So the simulator just has to cook up convincing looking values for $v_1, \ldots, v_N$ to send in step D3. With these values, the simulator computes $h_1, \ldots, h_N$ according to the rule in step D4, and "backpatches" the random oracle for $H_2$, setting $H_2(h_1, \ldots, h_N)$ to $h$.

For $1 \le i \le N$, the simulator computes $v_i$ as follows.

If $b_i = 0$, the simulator just chooses $v_i = r_i$, where $r_i$ is a random bit string of length $l$.

If $b_i = 1$, then the simulator just computes $v_i = (\psi_i, s_i'')$, where $s_i''$ is a random element of $G_1$, and $\psi_i$ is an escrow of an arbitrary group element under the condition $\kappa$.

This simulation is not perfect, since $\psi_i$ does not encrypt the right thing; however, since the adversary will by definition never obtain the decryption of $\psi_i$ under the condition $\kappa$, the security of the underlying escrow scheme, together with the pseudo-random property of $H_1$, implies that the adversary cannot detect the difference.

That completes the proof of Lemma 3.

**Remark 13** The restriction is step D2 that not all $b_i$ are zero is not strictly necessary. The only reason we make this restriction is so that the protocol has the following, stronger completeness property: if both $P$ and $V$ are honest, then for all $s$, $d$, and $\kappa$, with $\theta(s) = d$, $V$ accepts and outputs $\alpha$ where $\theta(D(\alpha, \kappa)) = d$.

**Remark 14** As in Remark 12, the random oracles representing $H_1$ and $H_2$ should be *independent* of any random oracles used in the underlying encryption scheme, or any other part of the system.

**Remark 15** The use of the random hash function model in this protocol is not essential. As already mentioned in the proof of Lemma 3, the only property required of $H_1$ is that it behave as a pseudo-random generator. As for $H_2$, it is evident from the proof that it could be implemented as a "trap door" or "chameleon" commitment scheme [BCC88, BKK90], based on the hardness of either the factoring or discrete logarithm problems.

### 6.3 An Example Implementation: Discrete Logarithms

Assume that $H_2$ has 160-bits of output, and that the input length $l$ of $H_1$ is 160 bits. Since the probability that an honest verifier is "cheated" in any one interaction is roughly $2^{-N}$, $N = 40$ should be sufficient for most applications. One could make the protocol non-interactive using the Fiat-Shamir heuristic, but then much larger values of $N$, say $N = 80$, would be required to avoid off-line attacks.

For our low-level encryption function, we take the OAEP encryption function of Bellare and Rogaway [BR94], based on the RSA problem. Assume a composite modulus of 1024 bits and an encryption exponent of 3. OAEP is secure against chosen ciphertext attacks in the random hash function model (although the proofs in [BR94] have to be adapted slightly to prove this).

Apropos DSS, for the discrete logarithm problem, assume the group $G_2$ is the subgroup of order $q$ in $\mathbf{Z}_p^*$, where $p$ is a 1024-bit prime, and $q$ is a 160-bit prime. Let $g$ be the given generator for $G_2$. In our notation, the group $G_1$ is the additive group $\mathbf{Z}_q$, and $\theta$ sends $a \in \mathbf{Z}_q$ to $g^a$.

The expected amount of data transmitted is between 3 and 4 KBytes.

Both $P$ and $V$ perform 40 160-bit exponentiations in $\mathbf{Z}_p^*$, all to the base $g$, plus the multiplications for OAEP. Using techniques of Lim and Lee [LL94], each party can do this using under 2000 modular multiplications. This bound already includes the pre-computation time for the base $g$. If $g$ is actually fixed for one of the parties, this bound can be reduced to about 1000 modular multiplications.

### 6.4 An Example Implementation: RSA Pre-images

Assume the same encryption function as above.

For the RSA pre-image problem, assume a 1024-bit composite modulus $M$ and encryption exponent $e$ with $(\phi(M), e) = 1$. In our notation, $G_1 = G_2 = \mathbf{Z}_M^*$, and $\theta(a) = a^e$. Typically, $e = 3$ or $e = 2^{16} + 1$. One could also take $e = 2$, in which case $G_2 = (\mathbf{Z}_m^*)^2$.

The expected amount of data transmitted is between 6 and 7 KBytes.

For $e = 3$, each party needs no more than 160 modular multiplications, and for $e = 2^{16} + 1$, this number is under 800.

## 7 The Fair Exchange Protocol

Now that we have all the necessary tools, we can easily describe our fair exchange protocol.

Suppose $A$ holds a signature $\sigma_A$ on message $m_A$ under public key $PK_A$, and $B$ holds a signature $\sigma_B$ on message $m_B$ under public key $PK_B$

We make use of our schemes for reducing signatures to homomorphic pre-images (§4). We also make use of an ordinary (non-verifiable) escrow scheme (§5) and a verifiable escrow scheme (§6); the public key for $T$ consists of a public key for the ordinary escrow scheme, and a public key for the verifiable escrow scheme.

We also make use of a one-way function $f$.

The third party $T$ maintains a set $S$ of tuples, initially empty, whose structure is described below. We describe the protocol assuming $B$ makes the first move.

**Protocol E**

E1 $B$ computes $\mathsf{reduce}(PK_B, m_B, \sigma_B) = (\theta_B, d_B, c_B, s_B)$, and sends $\theta_B, d_B, c_B$ to $A$.

E2 $A$ receives $\theta_B, d_B, c_B$ from $B$ and checks that

$$\mathsf{verify}(PK_B, m_B, \theta_B, d_B, c_B)$$

accepts. If this succeeds, $A$ chooses $r \in \mathsf{Domain}(f)$ at random, computes $v = f(r)$, and sends $B$ an ordinary escrow $\alpha$ of $\sigma_A$ with associated condition $(v, PK_A, m_A, \theta_B, d_B)$, along with $v$. Otherwise, $A$ quits.

E3 $B$ receives $\alpha, v$ from $A$. If this succeeds, $B$ and $A$ engage in the verifiable escrow protocol, with $B$ as prover and $A$ as verifier, so that $A$ obtains a verifiable escrow $\beta$ of $s_B$ with associated condition $(v, \alpha, PK_A, m_A, \theta_B, d_B)$. Otherwise, $B$ quits.

E4 If $A$ accepts the proof in E3, obtaining the escrow $\beta$, then $A$ sends $\sigma_A$ to $B$. Otherwise, $A$ invokes Protocol abort and then quits.

E5 $B$ receives $\sigma_A$ from $A$ and checks that $\sigma_A$ is a valid signature on $m_A$ with respect to $PK_A$. If this succeeds, $B$ sends $s_B$ to $A$, outputs $\sigma_A$, and quits. Otherwise, $B$ invokes Protocol B-resolve and then quits.

E6 $A$ receives $s_B$ from $B$ and checks that $\theta_B(s_B) = d_B$. If this succeeds, $A$ outputs $\mathsf{recover}(PK_B, m_B, \theta_B, c_B, s_B)$, and quits. Otherwise, $A$ invokes Protocol A-resolve and then quits.

**Protocol** abort

$A$ sends $r, \theta_B, d_B$ to $T$, who does the following:

let $v = f(r)$
if $(\mathsf{no\text{-}abort}, v) \in S$ then
  send $A$ the message "error"
else if $(\mathsf{deposit}, v, \theta_B, d_B, s_B) \in S$ (for some $s_B$) then
  send $s_B$ to $A$ (from which $A$ can recover $\sigma_B$)
else
  add $(\mathsf{abort}, v)$ to $S$
  send $A$ the message "exchange aborted"

**Protocol** B-resolve

$B$ sends $v, \alpha, PK_A, m_A, \theta_B, s_B$ to $T$, who does the following:

let $d_B = \theta_B(s_B)$
if $(\mathsf{abort}, v) \in S$ then
  send $B$ the message "exchange aborted"
else
  decrypt $\alpha$ subject to the condition $(v, PK_A, m_A, \theta_B, d_B)$
  check that the decryption is a valid signature $\sigma_A$ on $m_A$ with respect to $PK_A$
  if not
    send $B$ the message "error"
  else
    add $(\mathsf{deposit}, v, \theta_B, d_B, s_B)$ to $S$
    send $\sigma_A$ to $B$

**Protocol** A-resolve

$A$ sends $r, \alpha, \beta, PK_A, m_A, \theta_B, d_B$ to $T$, who does the following:

let $v = f(r)$
if $(\mathsf{abort}, v) \in S$
  send $A$ the message "error"
else
  add $(\mathsf{no\text{-}abort}, v)$ to $S$
  decrypt $\alpha$ subject to the condition $(v, PK_A, m_A, \theta_B, d_B)$
  check that the decryption is a valid signature on $m_A$ with respect to $PK_A$
  if not
    send $A$ the message "error"
  else
    decrypt $\beta$ subject to the condition $(v, \alpha, PK_A, m_A, \theta_B, d_B)$
    send result to $A$ (from which $A$ can recover $\sigma_B$)

**Remark 16** We remind the reader that in our system model (§3), $T$ processes requests atomically. This is required for correctness, although it can be relaxed to allow requests corresponding to different values $v$ to be processed concurrently.

**Remark 17** The value $v$ in the protocol acts as a sort of "transaction ID," and it also allows player $A$ to authenticate itself to $T$ (by presenting the pre-image of $v$ under $f$), so only $A$ can invoke sub-protocols $\mathsf{abort}$ and $\mathsf{A\text{-}resolve}$ with respect to the transaction ID $v$. In some protocols, it is necessary that both players contribute to the transaction ID, but it turns out not to be necessary here.

**Remark 18** The logic of the protocol dictates that for a given $v$, $S$ may contain either $(\mathsf{abort}, v)$ or $(\mathsf{no\text{-}abort}, v)$, but not both. Note that by invoking Protocol $\mathsf{B\text{-}resolve}$, tuples of the form $(\mathsf{deposit}, v, \ldots)$ may be placed in $S$ that have nothing to do with the data that $A$ expects; however, when $A$ invokes Protocol $\mathsf{abort}$, any such erroneous $\mathsf{deposit}$ tuples will not affect the outcome of the $\mathsf{abort}$ operation.

**Remark 19** It may not at first sight seem necessary to include $PK_A$ and $m_A$ in the condition attached to the escrow $\alpha$ created by player $A$; however, we need to do this in order to make the proof of security go through.

**Remark 20** In general, the public keys for the two escrow schemes should be independent. However, for the particular implementations we have suggested in this paper, it is easy to see that the same public key may be used for both.

**Remark 21** We can limit the degree of trust we place in $T$, so that the players need not reveal their messages or signatures to $T$. This can be done by the well known technique of blinding (see [FR97] for a detailed discussion of this). In the case of player $B$, whose secret is effectively a homomorphic pre-image, this can be achieved by having $B$ escrow a blinded version of this pre-image. In the case of player $A$, we would have to modify the protocol so that $A$ reduces his promise of a signature to a promise of a homomorphic pre-image as well, and then $A$ can escrow (again, non-verifiably) a blinded version of this pre-image just like $B$.

Our main result is the following.

**Theorem 1** *Assuming that $f$ is one-way, that the underlying signature, reduction, and escrow schemes are secure (in the random hash function model), then the above fair exchange protocol is secure (in the random hash function model).*

*Completeness.* Obvious.

*Fairness for $A$.* Consider an honest $A$ playing against a dishonest $B^*$. Let us say that $B^*$ wins the game if $B^*$ obtains $\sigma_A$, but $A$ does not obtain $\sigma_B$; otherwise, $B^*$ loses. Assume, by way of proof by contradiction, that $B^*$ has a non-negligible chance of winning this game.

Now, by the one-wayness of $f$, we may assume that $B^*$ cannot invoke sub-protocols $\mathsf{abort}$ and $\mathsf{A\text{-}resolve}$ with respect to the transaction ID $v$.

The technique we use to prove fairness is as follows. We truncate the game at various points, stopping the game early should certain conditions arise, and declaring $B^*$ the loser when this happens. Whenever the game is stopped early, we argue that $B^*$ would have lost anyway (with overwhelming probability) had the game continued from the point at which it was stopped, and so

$B^*$ must still have a non-negligible chance of winning this truncated game. We then show that we can effectively simulate the view of $B^*$ in this truncated game without knowing $\sigma_A$, which means that when $B^*$ wins, he has forged a signature, contradicting the assumed security of $A$'s signature scheme.

Now the details.

We truncate the game, so that if $B^*$ ever invokes Protocol B-resolve at some point after $A$ sends $\alpha$ to $B^*$, and the protocol has not been aborted by $A$, and the parameters $(v, \alpha, PK_A, m_a, \theta_B, s_B)$ supplied by $B^*$ are correct, we stop the game. $B^*$ would almost surely lose the game if it continued. The reason is that if this invocation of B-resolve were completed, then a correct deposit tuple would be placed in $S$. When the game continued, then either $A$ would obtain $\sigma_B$ directly by the normal run of the protocol, or would obtain it through Protocol abort (by virtue of the soundness of the reduction protocol), or through Protocol A-resolve (by virtue of the soundness of the reduction and verifiable escrow protocols).

Let us truncate the game further. Let us stop the game just before $A$ sends $\sigma_A$ to $B^*$ in step E4, if $A$ should reach that point. $B^*$ would almost surely lose the game. This is because either $A$ would receive $\sigma_B$ directly from $B^*$, or would obtain it through Protocol A-resolve (again, by virtue of the soundness of the reduction and verifiable escrow protocols).

We truncate the game even further. If $A$ ever executes Protocol A-resolve, we stop the game at that point. Again, if the game continued, $B^*$ would almost surely lose, since $A$ would obtain $\sigma_B$ in this invocation of A-resolve.

In this new truncated game, $A$'s escrow $\alpha$ is never decrypted with the correct condition, and $A$ never sends $\sigma_A$ in the clear. We now make a further modification to the game: instead of having $A$ escrow $\sigma_A$, we have it escrow garbage. By the security of the escrow scheme that $A$ used to create $\alpha$, it must be the case that $B^*$ still has a non-negligible chance of computing $\sigma_A$. We make one last modification to the game: we throw away the signature $\sigma_A$ used by $A$ in the protocol—we do not need it anymore. This does not change the game at all, and so $B^*$ can still compute $\sigma_A$ with non-negligible probability—but this represents a forgery, contradicting the security of $A$'s signature scheme.

*Fairness for B.* Consider an honest $B$ playing against a dishonest $A^*$. Let us say that $A^*$ wins the game if $A^*$ obtains $\sigma_B$, but $B$ does not obtain $\sigma_A$; otherwise, $A^*$ loses. Assume, by way of proof by contradiction, that $A^*$ has a non-negligible chance of winning this game.

As before, we truncate the game. This time, if $A^*$ invokes Protocol A-resolve at some point after $B$ has started the verifiable escrow protocol in step E3 of the main protocol, and Protocol A-resolve reaches the point where $\beta$ is decrypted, and if the condition under which $\beta$ is to be decrypted matches exactly the condition that $B$ attaches to its verifiable escrow in step E3 of the main protocol, we stop the game. If the game continued, $A^*$ would certainly lose. The reason for this is that since A-resolve reached this point, the escrow $\alpha$ has already been decrypted by $T$ who has verified that its contents are correct. Furthermore, a no-abort tuple is placed in $S$ that will prevent $A^*$ from ever aborting this transaction. Hence, $B$ will either obtain $\sigma_A$ directly from $A^*$, or will be able to obtain it via Protocol B-resolve.

We truncate the game further. We stop the game if $B$ invokes B-resolve, and that protocol reaches the point where the deposit tuple is to be added to $S$. Note that at this point, $T$ has decrypted $\alpha$ and verified the correctness of its contents. $A^*$ clearly will loses the game at this point, since $B$ will obtain $\sigma_A$.

We truncate the game even further, this time stopping the game just before $B$ sends $s_B$ in step E5. $A^*$ has already lost the game at this point, since $B$ does not send $s_B$ unless it already has obtained $\sigma_A$.

We now modify the game, "plugging in" the zero-knowledge simulator for the verifiable escrow protocol. Recall that this "plug and play" simulator is simply a machine that replaces various components of the verifiable escrow protocol, and does not require $s_B$ as an input. Thus we do not run into any subtle problems of protocol composition. Since the verifiable escrow is never decrypted with respect to the correct condition in this truncated game, $A^*$ must still win with non-negligible probability.

In this truncated, modified game, we see that $s_B$ is no longer necessary. So we modify the game even further, throwing away $s_B$. Now, since $A^*$ still has a non-negligible chance of winning this game, and hence a non-negligible chance of computing $\sigma_B$, we can use $A^*$ to break $B$'s reduction scheme—a contradiction.

**Remark 22** The role of "plug and play" simulatability in the above proof deserves further comment. Consider Game $\mathsf{C}$ in the definition of "plug and play" zero knowledge. In our application of this definition, the "adversary" in Game $\mathsf{C}$ is a quite complicated object, consisting not only of $A^*$, but also *all* parts of Protocol $\mathsf{E}$ that do not correspond to $D$ and $P$ in Game $\mathsf{C}$. In particular, this includes:

- the signing oracle for the key $SK_B$;
- all computations performed by $B$ outside of step $\mathsf{E3}$, including the computation of $\mathsf{reduce}$ in step $\mathsf{E1}$;
- all computations performed by $T$ other than the decryption of verifiable escrows.

We also define the bit output by the "adversary" in Game $\mathsf{C}$ so that it indicates if $A^*$ wins the truncated exchange game.

# 8 Exchanging Digital Data

In this section, we show how to adapt the protocol in §7 so as to allow the exchange of digital data, rather than digital signatures.

We consider two particular variants of this problem.

In the first variant, the "digital content" problem, suppose a customer wants to buy, say, some digital music from a vendor in exchange for an electronic check. The electronic check is a digital signature, and our protocol can deal with this already. However, we would not expect a general-purpose trusted third party for fair exchange to be able to ensure the correctness and quality of a string of bits that is supposed to represent a certain piece of digital music. Instead, we would expect a different third party to provide that service, guaranteeing the quality, and offering some kind of recourse in case this quality is not achieved.

The second variant we consider is certified e-mail. Here, one player is sending a message in return for a receipt on that message. The point, though, is that the receiver should not be able to selectively sign only messages that it wants to sign; rather, the receiver must effectively decide to sign a message without knowing its contents.

## 8.1 String Commitment

For both variants of this problem, a useful tool is *string commitment*. A string commitment scheme works as follows. A player commits to a bit string $x$ by computing $\mathcal{C} = F(x, s)$, where $s$ a random bit string, and $F$ is an efficiently computable function. A player opens a commitment by revealing $x$ and $s$. The two basic properties a commitment scheme should satisfy are:

- $\mathcal{C}$ should not leak information about $x$ (at least in a computational sense);

- it should be infeasible to open $\mathcal{C}$ in two different ways, i.e., to find $x, x', s, s'$, with $x \neq x'$, such that $F(x,s) = F(x',s')$.

For more on commitments, see [BCC88] We briefly discuss some simple, well known, and practical commitment schemes.

First of all, for long messages (like a digital music file), it is useful to use a hybrid commitment scheme, as follows. The commitment to a message $m$ consists of $\psi = E_K(m)$ and $\mathcal{C} = F(K,s)$, where $\psi$ is an encryption of $m$ under a symmetric encryption key $K$, and $\mathcal{C}$ is an ordinary commitment as above. To open a commitment, one must just reveal $K$ and $s$. We shall use such a hybrid commitment scheme in what follows.

One extremely simple and efficient way to implement $F$ is to simply use a cryptographic hash $H$, and set $F(K,s) = H(K,s)$. It is quite reasonable to conjecture that this is a secure commitment scheme, and is certainly justified in the random hash function model Alternatively, since $K$ is anyway random, it is not unreasonable to eliminate $s$ altogether from this scheme, and just commit to $K$ with $H(K)$.

Another way to implement $F$ is to define $F(K,s) = g_1^K g_2^s$, where $g_1$ and $g_2$ are random generators for some group $G$ of prime order $q$ such that the discrete logarithm problem is hard. We assume here that $K$ can be encoded as a number mod $q$, and $s$ is a random number mod $q$. The resulting scheme can be easily proved to be a secure commitment scheme assuming the discrete logarithm problem is hard. Alternatively, since $K$ is anyway random, we could instead commit to $K$ with $g^s$ for random $s$ mod $q$, where $g$ generates $G$, and then reveal $K$ by revealing $s$, from which $K$ is derived by computing $K = H(s)$. Again, this is justified in the random hash function model.

In both of the above discrete logarithm based commitment schemes, opening a commitment corresponds to revealing a homomorphic pre-image. In the first case, the homomorphism is from $\mathbf{Z}_q \times \mathbf{Z}_q$ to $G$, and in the second the homomorphism is from $\mathbf{Z}_q$ to $G$. This means that a commitment-opening can be verifiably escrowed using the protocol in §6.

### 8.2 The digital content problem

Now consider the "digital content" problem. Let $m$ be, say, a certain digital music file. Then the music vendor can produce the hybrid commitment $\psi = E_K(m)$, $\mathcal{C} = F(K,s)$ as above, and has a third party sign a message consisting of $\psi$, $\mathcal{C}$, and a description of the musical content. This signature acts as a guarantee that any opening of this commitment is a copy of the indicated music.

Now suppose customer wants to buy this digital music from a vendor, in exchange for an electronic check. The vendor gives the customer $\psi$ and $\mathcal{C}$, along with the signature from the third party guaranteeing the quality of the content.

The items to be exchanged in the fair exchange protocol are the electronic check in exchange for the opening $(K,s)$ of $\mathcal{C}$. We can trivially adapt Protocol $\mathsf{E}$ in §7 to this problem. We can have the vendor play the role of player $A$, and the customer the role of player $B$. We sketch the differences:

- In step $\mathsf{E2}$, $A$ creates an escrow $\alpha$ of $(K,s)$ with attached condition $(v, \mathcal{C}, \theta_B, d_B)$.

- In step $\mathsf{E3}$, $B$ creates the verifiable escrow $\beta$ of $s_B$ with attached condition $(v, \alpha, \mathcal{C}, \theta_B, d_B)$.

- In step $\mathsf{E4}$, $A$ sends $(K,s)$ to $B$.

- In step $\mathsf{E5}$, $B$ receives $(K,s)$ and checks that $F(K,s) = \mathcal{C}$.

- In protocols A-resolve and B-resolve, the players send $\mathcal{C}$ instead of $PK_A, m_A$, and when $T$ decrypts $\alpha$, it checks that the contents are of the form $(K, s)$ where $F(K, s) = \mathcal{C}$.

Note that if we want to allow various commitment schemes, then the protocol must be modified to include a description of the scheme in the conditions attached to the escrow and in the parameters passed to $T$.

Note that because only player $B$ needs to compute a verifiable escrow, it is most convenient to have the player who escrows the commitment-opening play the role of player $A$, since this allows one to use an arbitrary commitment scheme with no particular algebraic structure. However, if both players want to exchange commitment-openings, then one of these two commitment schemes must be implemented as a one-way group homomorphism, like the discrete logarithm based commitment schemes mentioned above, so as to allow an efficient verifiable escrow of the commitment-opening using the protocol in §6.

**Remark 23** It should be noted here that Protocol E can be easily adapted so as to allow the exchange of *any* secret satisfying a particular, efficiently computable predicate, in exchange for a homomorphic pre-image—or any secret that can be reduced to a homomorphic pre-image.

### 8.3 Certified e-mail

Now consider the certified e-mail problem. Here the sender begins by sending the receiver $\psi = E_K(m)$ and $\mathcal{C} = F(K, s)$, where $m$ is the actual message to be sent. Now the sender and receiver engage in the modification to protocol E sketched above, where the sender plays the role of player $A$, the receiver the role of player $B$, and the message that $B$ agrees to sign contains $\psi$ and $\mathcal{C}$. Note, however, that such a signature has to be interpreted in a special way: it is only considered to be a valid signature when accompanied by $(K, s)$ such that $F(K, s) = \mathcal{C}$, and the message that is effectively signed is the decryption of $\psi$ under the key $K$.

Thus, in this solution to the certified e-mail problem, the receipts that are generated are of a non-standard form, dictated by the fair exchange protocol. Because of this, one might forego the use of verifiable escrow, and combine elements of the above protocol with elements of Protocol F in §10 to obtain a somewhat more efficient protocol. This is a straightforward exercise that we leave to the reader.

Note that in these solutions to the "digital content" and certified e-mail problems, the third party $T$ in the exchange protocol obtains no useful information about the digital data. It only obtains $K$, and without $\psi$, this gives $T$ no information about the digital data itself. Thus, we do not need to sacrifice the privacy of our data by using the service provided by $T$.

## 9 Verifiable Escrow with Off-Line Coupons

In this section, we describe a more efficient solution to the verifiable escrow problem. Since we are already using a trusted third party for the fair exchange protocol, we can use a trusted third party to generate certified "coupons" that can be used to efficiently generate and validate the verifiable escrows that we need. These coupons can be obtained in bulk and off line, before performing any exchanges.

Assume we have a group homomorphism $\theta : G_1 \to G_2$. We assume that the trusted third party $T$ has a public signing key and a public key for a semantically secure encryption scheme.

A *coupon* is a tuple of the form $(\theta, d', \psi, PK)$, where

- $\theta$ is a description of the homomorphism, including $G_1$ and $G_2$;
- $d' \in G_2$;
- $\psi$ is an encryption under $T$'s public key of $s' \in G_1$ such that $\theta(s') = d'$;
- $PK$ is a public key for some secure signature scheme (with corresponding secret key $SK$);

together with $T$'s signature on this tuple.

To obtain a coupon, a player sends a request for a coupon to $T$, including $\theta$. Upon receipt of this request, $T$ generates $s' \in G_1$ at random, along with $(PK, SK)$, and then constructs the coupon as above, sending the coupon, $s'$, and $SK$ to the player. We assume this transaction takes place using a secure channel.

Now suppose a player wants to construct a verifiable escrow of a homomorphic pre-image $s \in G_1$ of $d \in G_2$ with attached condition $\kappa$. The verifiable escrow consists of a coupon as above, together with $s'' = s + s' \in G_1$, and a signature on $\kappa$ using the key $SK$.

To verify such an escrow, one checks that $\theta(s'') = d + d'$, verifies $T$'s signature on the coupon using $T$'s public key, and verifies the signature on $\kappa$ using the key $PK$. Clearly, if one obtains a decryption of $\psi$, one can readily compute $s = s'' - s'$.

Note that to maintain security, each verifiable escrow should use a fresh coupon, i.e., a coupon should be used once and then thrown away.

Our fair exchange protocol in §7 can be easily modified to accommodate such a coupon-based verifiable escrow. We leave the details to the reader.

There are several variants that one might consider. First, a player might generate $(SK, PK)$ and $s'$ on his own, instead of having $T$ do this. Second, the public key encryption function might be replaced with a symmetric-key cipher, whose key is known to $T$. Or alternatively, $T$ could just store $s'$ and later reveal it, but this is perhaps impractical.

Note that in our construction, a coupon can be obtained by a player in advance of knowing the type of item it expects to receive in exchange for the item it is giving: the condition $\kappa$ can be attached to the coupon at the last minute, when the coupon is actually used. Construction of the coupon does depend, however, on the type of item the player is giving. But in many practical situations, this is quite acceptable. For example, a player who wishes to purchase items using his electronic checks can obtain many coupons in advance corresponding to the signature scheme used for his electronic checks. A travel agent selling airline tickets can obtain coupons corresponding to the signature scheme used to sign electronic airline tickets. This requires that the homomorphism used for the reduction scheme associated with the signature scheme depends only on the public key of the signature scheme, and not on a particular signature. This is the reason for the alternative, and somewhat more complicated reduction scheme we gave for DSS in §4.3.

## 10 An Accountable Protocol for Contract Signing

In this section we consider the contract-signing problem: two players wish to sign a contract $m$ in such a way that either each player obtains the other's signature, or neither player does.

We could, of course, use the protocol in §7. However, if we allow ourselves the flexibility to prescribe the form of the signatures, we can obtain a protocol that is much more efficient, and that achieves another important property lacking in the previous protocols: accountability.

By *accountability*, we mean that if the trusted third party $T$ misbehaves, then this can be *proven*. Suppose that after executing the protocol, $A$ does not obtain a valid contract. If $T$ does not misbehave, then $B$ will not have a valid contract either. However, if $B$ *does* have a valid contract, and ever tries to enforce it in court, then the contract that $B$ presents in court, together with information obtained by $A$ during the execution of the protocol, can be used to *prove* that $T$ misbehaved. At this point, the contract is ruled invalid, and both $A$ and $B$ sue $T$ for misbehaving. Of course, it must be infeasible for $A$ and $B$ to collude to somehow *frame* $T$ if $T$ does not misbehave.

Note that we can combine such an accountable contract-signing protocol with our general exchange protocols. If $A$ wants to send $B$ an electronic check in exchange for an electronic airline ticket, then in stage 1, $A$ and $B$ sign a contract to this effect using an accountable contract-signing protocol. In stage 2 they use our more general (but unaccountable) fair exchange protocol to actually exchange the items. If something goes wrong, both parties still have legal recourse: either the contract can be enforced via legal action, or $T$ can be sued for misbehaving in stage 1. Note that even with the contract from stage 1, it is still worthwhile to use a fair exchange protocol in stage 2, as one generally wants to avoid the time and expense of going to court to enforce a contract.

### 10.1 The Protocol

At a high level, the protocol works by first having the two players exchange signed messages that act as "pre-contracts," which are themselves not considered to be valid contracts. After having exchanged pre-contracts, the players exchange actual contracts. If anything goes wrong, a player either can request that $T$ abort or resolve the transaction, where in this setting, $T$ resolves a transaction by *converting* a pair of pre-contracts to a valid contract by affixing a signature of its own to the pair of pre-contracts.

We now describe our contract-signing protocol in greater detail. For $X \in \{A, B, T\}$, we denote by $[\alpha]_X$ the string $\alpha$, concatenated with a signature of $\alpha$ under $X$'s public key. $T$ maintains a set $S$ of messages.

For simplicity, we assume that an honest player takes steps to ensure that it never engages in the contract signing protocol more than once with the same message. This can be easily achieved by attaching a *nonce* to the original message.

#### Protocol F

F1 $A$ sends $\alpha = [m, A, B, T]_A$ to $B$.

F2 $B$ receives and verifies the signature $\alpha$ from $A$. If this succeeds, $B$ sends $\beta = [m, A, B, T]_B$ to $A$. Otherwise, $B$ quits.

F3 $A$ receives and verifies the signature $\beta$ from $B$. If this succeeds, $A$ sends $\alpha' = [m, A, B]_A$ to $B$. Otherwise, $A$ invokes Protocol abort and then quits.

F4 $B$ receives and verifies the signature $\alpha'$ from $A$. If this succeeds, $B$ sends $\beta' = [m, A, B]_B$ to $A$, outputs $(\alpha', \beta')$, and quits. Otherwise, $B$ invokes Protocol resolve and then quits.

F5 $A$ receives and verifies the signature $\beta'$ from $B$. If this succeeds, $A$ outputs $(\alpha', \beta')$ and quits. Otherwise, $A$ invokes Protocol resolve and then quits.

We summarize the message flows:

$A \to B: \ [m, A, B, T]_A$

$B \to A: \ [m, A, B, T]_B$

$A \to B: \ [m, A, B]_A$

$B \to A: \ [m, A, B]_B$

### Protocol $\mathsf{abort}$

$A$ sends to $T$ the message $[m, A, B, \mathsf{abort}]_A$. $T$ verifies the signature, and then checks if there is a message of the form $[[m, A, B, T]_A, [m, A, B, T]_B]_T$ in $S$. If so, $T$ returns this message to $A$. Otherwise, $T$ returns $[[m, A, B, \mathsf{abort}]_A]_T$ to $A$, and also stores this message in $S$.

### Protocol $\mathsf{resolve}$

A player sends to $T$ the message $([m, A, B, T]_A, [m, A, B, T]_B)$. $T$ verifies the signatures, and checks if there is a message of the form $[[m, A, B, \mathsf{abort}]_A]_T$ in $S$. If so, $T$ returns this message to the player. Otherwise, $T$ generates $[[m, A, B, T]_A, [m, A, B, T]_B]_T$, sends this message to the player and stores it in $S$.

A *valid contract* on $m$ between $A$ and $B$ is of the form

$$([m, A, B]_A, [m, A, B]_B)$$

or

$$[[m, A, B, T]_A, [m, A, B, T]_B]_T,$$

or if it is the above form with the roles of $A$ and $B$ reversed.

If ever a pair of strings of the form

$$([[m, A, B, \mathsf{abort}]_A]_T, \ [[m, A, B, T]_A, [m, A, B, T]_B]_T)$$

is produced, $T$ will be judged to have *cheated*.

It is assumed that whenever $T$ is given a query of the form

$$([m, A, B, T]_A, [m, A, B, T]_B)$$

or

$$[m, A, B, \mathsf{abort}]_A,$$

then $T$ can be *compelled* to give a response of the form

$$[[m, A, B, T]_A, [m, A, B, T]_B]_T$$

or

$$[[m, A, B, \mathsf{abort}]_A]_T.$$

Such an assumption is reasonable, as $T$ is supposed to be a well-known entity providing a guaranteed service; if $T$ is not able to respond within a reasonable amount of time with a valid response, it could be held liable for failure to provide service.

### 10.2 Security analysis

Security for an honest player means the following. Suppose an adversary has corrupted the other player and $T$. Then at the end of the protocol, if the adversary obtains a valid contract, but the honest player does not, then the adversary's valid contract, together with information obtained by the honest player, can be used to prove that $T$ cheated.

Security for $T$ means that if $T$ is honest, an adversary cannot produce evidence that $T$ cheated (thereby framing $T$).

*Security for $A$.* Suppose $A$ is honest and does not obtain a valid contract at the end of the protocol.

Suppose that $A$ terminated the protocol via the abort protocol. Then $A$ must have a string of the form $[[m, A, B, \mathsf{abort}]_A]_T$. If the adversary has a valid contract, it must be of the form $[[m, A, B, T]_A, [m, A, B, T]_B]_T$. These two items can thus be used to prove that $T$ cheated.

$A$ could not have terminated using Protocol resolve, since $A$ never generated $[m, A, B, \mathsf{abort}]_A]$, and hence the only possible response from $T$ in this case is a valid contract $[[m, A, B, T]_A, [m, A, B, T]_B]_T$.

*Security for $B$.* Suppose that $B$ is honest and does not obtain a valid contract.

If $B$ had simply quit the protocol in F2, the adversary can not possibly have a valid contract, as $B$ never signed anything.

If $B$ terminated via Protocol resolve, then $B$ must have a string of the form $[[m, A, B, \mathsf{abort}]_A]_T$. If the adversary has a valid contract, it must be of the form $[[m, A, B, T]_A, [m, A, B, T]_B]_T$. These two items can be used to prove that $T$ cheated.

*Security for $T$.* It is clear from the definitions of protocols abort and resolve that an uncorrupted $T$ will never output both $[[m, A, B, T]_A, [m, A, B, T]_B]_T$ and $[[m, A, B, \mathsf{abort}]_A]_T$.

**Remark 24** The technique used here of generating a pre-contract which can be converted to a contract can also be applied to the certified e-mail protocol in §8.3 to obtain a simpler, and more efficient protocol. Indeed, since the receipts generated in the certified e-mail protocol are already of a form that is determined by the protocol, there is no reason not to do this.

**Remark 25** We stress that a signed "abort message" from $T$ by itself has absolutely no significance, and one should not attempt to attach any significance to it. Indeed, $A$ and $B$ could run the protocol through to completion, both obtaining a contract, and then at a later time, $A$ could "abort" the transaction, obtaining a signed "abort message" from $T$. This is an amusing, but otherwise *completely irrelevant* characteristic of the protocol. Similarly, two players could end up with equivalent, but syntactically different contracts. Again, this may be interesting, but it is *completely irrelevant*.

## 11 Conclusion

We close by discussing two directions for future research related to this topic.

First, it would be nice to find a more efficient method of performing verifiable escrow, and in particular avoid the "cut and choose" method that we use. Even a less general or less secure, but more efficient method would be of interest.

Second, the fact that our trusted third party is off line means that we could afford to implement it as a distributed, fault-tolerant system, making it highly secure using somewhat more expensive cryptographic techniques, but eliminating the single point of failure. In future work, we intend to design and implement such a system. The protocols will require some modification. In addition, a threshold decryption scheme secure against chosen ciphertext attack is required; the threshold decryption scheme in [SG98] is the only available scheme that is truly practical and provably secure (in the random hash function model). For contract signing, we also would need a threshold signature scheme; the threshold signature scheme in [Sho99] is the only available scheme that is truly practical and provably secure (again, in the random hash function model). Also, it is apparently necessary to solve a sort of Byzantine agreement problem among the distributed servers, who must decide whether a particular transaction is to be "aborted" or "resolved"; Byzantine agreement has been extensively studied, but the new protocol in [CKS99] is a particularly practical one that can be rigorously analyzed and works in an asynchronous network.

## Acknowledgments

We would like to thank Ivan Damgard for several comments on an earlier version of this paper that we have incorporated.

## References

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *4th ACM Conference on Computer and Communication Security*, pages 6–17, 1997.

[ASW98] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology–Eurocrypt '98*, 1998.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Compter Systems*, 8:18–36, 1990.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, 1988.

[BDM98] F. Bao, R. H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line TTP. In *IEEE Symp. Security and Privacy*, pages 77–85, 1998.

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122, 1991.

[BG96] M. Bellare and S. Goldwasser. Encapsulated key escrow. Preprint, 1996.

[BKK90] J. Boyer, S. A. Kurtz, and M. W. Krentel. A discrete logarithm implementation of perfect zero-knowledge blobs. *J. Cryptology*, 2(2):63–76, 1990.

[BP90] H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers and Security*, 9:715–721, 1990.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Crypto '94*, pages 92–111, 1994.

[Bra93] S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology–Crypto '93*, pages 302–318, 1993.

[CKS99] C. Cachin, K. Kursawe, and V. Shoup. Practical asynchronous Byzantine agreement using cryptography. In preparation, 1999.

[CP92] D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology–Crypto '92*, pages 89–105, 1992.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology–Crypto '98*, pages 13–25, 1998.

[CTS95] B. Cox, J. D. Tygar, and M. Sirbu. NetBill security and transaction protocol. In *First USENIX Workshop on Electronic Commerce*, pages 77–88, 1995.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[DGLW96] R. H. Deng, L. Gong, A. A. Lazar, and W. Wang. Practical protocols for certified electronic mail. *J. of Network and Systems Management*, 4(3), 1996.

[FR95] M. K. Franklin and M. K. Reiter. Verifiable signature sharing. In *Advances in Cryptology–Eurocrypt '95*, pages 50–63, 1995.

[FR97] M. K. Franklin and M. K. Reiter. Fair exchange with a semi-trusted third party. In *4th ACM Conference on Computer and Communications Security*, pages 1–5, 1997.

[FS87] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology–Crypto '86, Springer LNCS 263*, pages 186–194, 1987.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17:281–308, 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18:186–208, 1989.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38:692–729, 1991.

[GO94] O. Goldreich and Y. Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7:1–32, 1994.

[GQ90] L. Guillou and J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology–Crypto '88, Springer LNCS 403*, pages 216–231, 1990.

[Jak92] M. Jakobsson. Reducing costs in identification protocols. Manuscript available at `http://www-cse.ucsd.edu/users/markus`, 1992.

[Kra93] D. W. Kravitz. Digital signature algorithm, 1993. U. S. Patent No. 5,231,668.

[LL94] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology–Crypto '94*, pages 95–107, 1994.

[Mic97] S. Micali. Certified e-mail with invisible post offices. Manuscript (presented at the *1997 RSA Security Conference*), 1997.

[OS91] H. Ong and C. Schnorr. Fast signature generation with a Fiat Shamir-like scheme. In *Advances in Cryptology–Eurocrypt '90, Springer LNCS 473*, pages 432–440, 1991.

[RS91] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology–Crypto '91*, pages 433–444, 1991.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, pages 120–126, 1978.

[Sch91] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.

[Sch96] C. Schnorr. Security of $2^t$-root identification and signatures. In *Advances in Cryptology–Crypto '96, Springer LNCS 1109*, pages 143–156, 1996.

[Sch97] C. Schnorr. Erratum: Security of $2^t$-root identification and signatures. In *Advances in Cryptology–Crypto '97, Springer LNCS 1294*, 1997.

[SG98] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology–Eurocrypt '98*, 1998.

[Sho96] V. Shoup. On the security of a practical identification scheme. In *Advances in Cryptology–Eurocrypt '96*, 1996. To appear, *J. Cryptology*.

[Sho99] V. Shoup. Practical thresold signatures. IBM Research Report RZ 3121, 1999.

[Sta96] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology–Eurocrypt '96*, pages 190–199, 1996.

[YY98] A. Young and M. Yung. Auto-recoverable auto-certifiable cryptosystems. In *Advances in Cryptology–Eurocrypt '98*, 1998.