Securing Threshold Cryptosystems against Chosen Ciphertext Attack

Victor Shoup* Rosario Gennaro**

Abstract. For the most compelling applications of threshold cryptosystems, security against chosen ciphertext attack seems to be a requirement. However, there appear to be no practical threshold cryptosystems in the literature that are provably chosen-ciphertext secure, even in the idealized random hash function model. The contribution of this paper is to present two very practical threshold cryptosystems, and to prove that they are secure against chosen ciphertext attack in the random hash function model.

1 Introduction

In this paper, we consider the problem of designing threshold cryptosystems that are secure against chosen ciphertext attack. Our goal is to design a practical scheme, and provide strong evidence that it cannot be broken.

Known standard (i.e., non-threshold) cryptosystems that are provably secure against chosen ciphertext attacks (under standard intractability assumptions) are unfortunately quite impractical, and thus only of theoretical interest. We cannot expect to do any better for threshold cryptosystems.

However, practical non-threshold schemes have been devised and proven secure in the so-called *random hash function*, or *random oracle*, model. This approach, first used by Fiat and Shamir [15], and later given a more rigorous treatment by Bellare and Rogaway [2], works as follows. Suppose that a cryptographic scheme that makes use of hash functions can be proven secure under standard intractability assumptions, but in an idealized model of computation where the hash functions are replaced by "black boxes" that output random strings. Then the basic tenet of the random hash function approach is to view this as "strong evidence" that the scheme is secure. See [2] for further discussion of the random hash function model.

Even though the most compelling applications of threshold cryptosystems seem to require chosen-ciphertext security, there appear to be no practical threshold cryptosystems in the literature that are provably secure, even in the random hash function model. Our main contribution is to present and analyze two such schemes. The first scheme, which we call TDH1 (for Threshold Diffie-Hellman), is secure assuming the hardness of the Diffie-Hellman problem [13]. The second

* IBM Zurich Research Lab, Saumerstrasse 4, CH-8803 Rueschlikon, Switzerland. sho@zurich.ibm.com

** IBM T. J. Watson Research Center, P.O.Box 704, Yorktown Heights NY 10598, USA. rosario@watson.ibm.com

scheme, TDH2, is secure under the stronger assumption of the hardness of the Diffie-Hellman *decision* problem, but is more efficient than TDH1.

1.1 Threshold cryptosystem and chosen ciphertext attack

In a threshold cryptosystem there is a single public encryption key, but the corresponding private decryption key is shared among a set of, say, n decryption servers in such a way that a threshold, say, k of them must cooperate to decrypt a message. The servers output decryption shares that can be combined to obtain the cleartext.

Security against chosen ciphertext attack intuitively means the following. A message m is encrypted, yielding a ciphertext c, and an adversary is allowed to feed the decryption system arbitrary ciphertexts c'—different from, but perhaps related to c—obtaining the corresponding decryption. Security in this scenario means that the adversary should gain no useful information about the message m.

In a threshold cryptosystem, the adversary has more power. Firstly, the adversary may corrupt a subset (of size at most $k-1$) of the decryption servers; in particular, the adversary knows their shares of the secret key. Secondly, and perhaps more importantly, the adversary can obtain not only the the decryption of a given ciphertext, but also the decryption shares generated by *all* of the servers. There is clearly no way to prevent this, since even if it is the responsibility of one of the servers to combine shares, this server may be corrupt and collaborate with the adversary.

1.2 Motivation

One of the main motivations for a threshold cryptosystem is that it allows one to construct a third-party decryption service in a distributed, secure, fault-tolerant fashion, without a significant increase in the size of or the cost of creating a ciphertext vis-à-vis a standard cryptosystem. To be at all useful, the third party should not decrypt everything that comes its way and give it to just anybody, but should implement some kind of useful policy. To implement such a policy securely, in addition to chosen-ciphertext security, one needs an additional facility: the ability to attach a *label* to the ciphertext during the encryption process. Such a label is a bit string that contains information that can be used by the third party to determine if the decryption request is authorized, according to its policy. Formally, one should think of the label as being a part of the ciphertext, so that changing the label changes the ciphertext; security against chosen ciphertext attack would then imply, in particular, that one cannot subvert the third party's policy by simply swapping labels.

Perhaps the most obvious example of this is *key recovery*. Here, two parties who wish to communicate generate a session key, and encrypt the session key under a third party's public key. The party that creates the encryption attaches a label containing the identities of the two parties, and the current time. This labeled ciphertext is sent along the wire, along with the encrypted conversation.

A law enforcement agency may be authorized via a court order to tap the line, and request that the third party decrypt the ciphertext containing the session key. To protect individual privacy, the court order specifies to whom the wiretap applies and a time interval. To enforce this policy, the third party only decrypts a ciphertext if the information in its label is consistent with the given court order.

Another example is the recent work of [1] on *fair exchange*, where an "off line" trusted third party is used to enforce fairness.

A label might also contain the identity or public key of the intended recipient, allowing the decryption service to direct the cleartext to that recipient only.

The usefulness of labeled ciphertext was already observed by Lim and Lee [18] (who called it an *indicator*). In a non-threshold cryptosystem, labeled ciphertexts can be implemented by simply embedding a hash of the label in the cleartext before encrypting. The decryption service is given a ciphertext and a label, computes the cleartext, and compares the value of the embedded hash with the hash of the given label. If these match, and the decryption policy authorizes the given label, then the cleartext is released. If the underlying cryptosystem is secure against chosen ciphertext attack, then so too will be the cryptosystem with labeled ciphertexts. This implementation is not suitable for threshold cryptosystems since the attacker who is mounting the chosen ciphertext attack may be cooperating with some of the decryption servers. Those servers see the decrypted labeled plaintext before it is output, thus it is too late at that point to check if the label is correct, since the attacker has already seen the result of the decryption operation.

1.3 Outline of the paper

In §2, we review relevant background and related work. In this section, we also sketch a simpler threshold cryptosystem, which we call TDH0, which may indeed be secure, but for which it seems difficult to obtain a security proof under standard cryptographic assumptions, even in the random hash function model. In §3, we define what we believe is a simple yet adequate formal security model. In §4, we briefly discuss the basic technical tools we use in the design and analysis of our cryptosystems, TDH1 and TDH2, which we describe and analyzed in 5 and 6, respectively. In §7 we discuss some implementation issues, and we conclude with some final remarks and open questions in §8.

2 Background and Related Work

2.1 Chosen ciphertext attack

Impractical but provable schemes. In the context of standard (i.e., non-threshold) cryptosystems, provably secure cryptosystems secure against chosen ciphertext attack were given by Naor and Yung [20], Rackoff and Simon [23], Dolev, Dwork, and Naor [14], and De Santis and Persiano [10]. Unfortunately, all known provably secure schemes rely on theoretical constructions of non-interactive zero-knowledge proofs [4], and as such are quite impractical.

Practical schemes. Again in the context of non-threshold cryptosystems, practical cryptosystems intended to be secure against chosen ciphertext attack were proposed by Damgard [8], Zheng and Seberry [28], and Bellare and Rogaway [2,3]. The schemes in [28,2,3] are all known to be chosen-ciphertext secure in the random hash function model.

2.2 Threshold cryptosystems

Threshold cryptosystems are part of a general approach known as *threshold cryptography*, introduced by Boyd [5], Desmedt [11], and Desmedt and Frankel [12]. In particular, in [12], a threshold cryptosystem based on the Diffie-Hellman problem is presented. The techniques developed later by De Santis *et al* [9] yield a corresponding system based on RSA [24]. These schemes can be shown to withstand chosen plaintext attack, but they are not known to withstand chosen ciphertext attack.

2.3 Why isn't it trivial to secure a threshold cryptosystem against chosen ciphertext attack?

Our first observation is that none of the *practical* schemes mentioned above can be readily transformed into threshold schemes that are chosen-ciphertext secure. To see why, consider the scheme in [2], which is representative. This scheme uses a trapdoor permutation f and hash functions G and H; to encrypt a message m, a random r in the domain of f is chosen, and the ciphertext is $(f(r), m \oplus G(r), H(r,m))$. The output length of G is equal to that of m, and the output length of H is large enough to make it difficult to find collisions. Given a ciphertext (s,c,v), the decryption algorithm computes $r = f^{-1}(s)$, $m = G(r) \oplus c$, and $v' = H(r,m)$. If $v' = v$, it outputs m, and otherwise "?".

The proof of security in the random hash function model relies in a critical way on the fact that the decryption algorithm makes the "validity test" $v = v'$ before generating an output.

Now consider turning this into a threshold scheme, and assume we can effectively share the trapdoor. The problem is that the above validity test cannot be performed until *after* the individual shares of $f^{-1}(s)$ are generated and then combined. As mentioned above, we must assume that the adversary can see these shares, making the validity test pointless, and giving the adversary the ability to invert f at chosen points. This destroys any hope of proving security using current techniques.

The above difficulty was noted by Lim and Lee [18], who observed that a publicly checkable validity test would be useful in this regard. Lim and Lee proposed two practical systems based on this observation; however, both schemes were subsequently broken by Frankel and Yung [16].

Interestingly, one can readily convert all of the *impractical* schemes mentioned above into secure (but impractical) threshold schemes. It is instructive to see why this is so. All of these schemes use a publicly checkable validity test, which is

essentially a non-interactive zero-knowledge proof of knowledge of the plaintext. The key to the proof of security is that one can simulate the adversary's view with a simulator that has a trapdoor that allows it to extract the plaintext from the given proof of knowledge in a decryption request, thus allowing the simulator to respond correctly to the request. Assuming the underlying decryption function can be effectively shared, such a scheme can then be transformed into a threshold scheme where each decryption server performs the validity test *before* generating a decryption share.

2.4 The TDH0 Cryptosystem

What makes the above approach impractical is the proof of knowledge. An obvious thing to try is to design an appropriate zero-knowledge proof of knowledge of the plaintext in the random hash function model that is more practical.

One obvious "solution" is the following threshold cryptosystem, based on the Diffie-Hellman problem, which we call TDH0. Say we have a group G of prime order q with generator g, a hash function H, and a public key $h = g^x$. To encrypt a message m, we choose $r \in \mathbf{Z}_q$ at random, and compute $u = g^r$ and $c = H(h^r) \oplus m$. The ciphertext consists of u, c, and a non-interactive proof of knowledge of $\log_g u$. It is straightforward to share the secret key, and a decryption server only generates a decryption share if the proof of knowledge is valid.

For the non-interactive proof of knowledge, we could use Schnorr's [25] signature scheme with "public key" u and "private key" r.

Intuitively, this strategy makes sense, and TDH0 may very well be secure, although it does not seem possible to prove that it is secure using known techniques, under standard cryptographic assumptions, *even in the random hash function model.*

The problem is a bit subtle. In the random hash function model, Schnorr's signature scheme is indeed a proof of knowledge, but the corresponding knowledge extractor does not operate "on line"—it must rewind the adversary many times, each time running it forward with different random outputs for the hash function used in the signature scheme. This type of "rewinding" extractor is adequate to prove that the corresponding signature scheme is secure in the random hash function model, but appears to be inadequate to prove the security of TDH0. The reason is that a simulator for TDH0 must respond to many decryption requests on-line, and all of this rewinding—possibly "undoing" previous decryption requests which recursively leads to more rewinding—results in a blow-up in the running time of the simulator that is exponential in the number of decryption requests. A similar phenomenon was observed by Pointcheval and Stern [22] in the context of blind digital signatures.

One could circumvent all this by straightaway assuming an on-line knowledge extractor; that is, we simply *assume* that any algorithm that can create create a valid proof of knowledge can be transformed into an algorithm that simultaneously outputs a corresponding witness. A similar type of assumption is made by Damgard [8] and Zheng and Seberry [28]. This type of assumption, however, is

not very acceptable: it is completely nonstandard, and it is not at all clear how it is related to any kind of intractability assumption.

3 A Formal Security Model

A k out of n threshold cryptosystem consists of the following components:

- A *key generation algorithm* that takes as input a security parameter, the number n of decryption servers, and the threshold parameter k; it outputs a *public key* PK, and a list $SK_1, \ldots, SK_n$ of private keys.
- An *encryption algorithm* that takes as input the public key PK and a cleartext m, and a label L, and outputs a ciphertext c.
- A *decryption algorithm* that takes as input the public key PK, an index $1 \leq i \leq n$, the private key SK_i, a ciphertext c, and a label L, and outputs a *decryption share* σ_i.
- A *recovery algorithm* that takes as input the public key PK, a ciphertext c, a label L, and a list $\sigma_1, \ldots, \sigma_n$ of decryption shares, and outputs a cleartext m.

Operation of the cryptosystem runs as follows. There is a trusted dealer and a set $P_1, \ldots, P_n$ of decryption servers. Under certain conditions the presence of a trusted dealer can be eliminated [21].

In an *initialization phase*, the dealer is run, creating the public and private keys. For $1 \leq i \leq n$, the public key PK and private key SK_i are given to server P_i.

A user who wants to encrypt a message can run the encryption algorithm, using the public key.

A user who wants to decrypt a ciphertext gives the ciphertext to each server P_i, who each generate a decryption share σ_i. These shares are then combined using the recovery algorithm to obtain the cleartext.

Note that the above model of a threshold cryptosystem is not the most general possible: the need for a trusted dealer may be relaxed, and decryption may involve a more general distributed computation. We do not discuss these possible variations in this paper.

Any practical k out of n cryptosystem should be *robust*, i.e., it should be able to tolerate the presence of an adversary that tries to hinder the recovery process. When a ciphertext c is decrypted, a list of corresponding decryption shares $\sigma_1, \ldots, \sigma_n$ is produced. The scheme is robust if the recovery algorithm recovers the plaintext m (at least with overwhelming probability) even if at most $k-1$ of the decryption shares are incorrect. Robust threshold cryptosystems are presented in [21,17] and we follow their general approach in order to make our schemes robust.

To define *security against chosen ciphertext attack*, we consider the following game played against an adversary.

Game A

A1 The adversary chooses to corrupt a fixed set of $k-1$ servers.
A2 The key generation algorithm is run. The private keys of the corrupted servers are given to the adversary, while the other private keys are given to the uncorrupted servers, and kept secret from the adversary. The adversary of course receives the public key as well.
A3 The adversary interacts with the uncorrupted encryption servers in an arbitrary fashion, feeding them ciphertext/label pairs, and obtaining decryption shares.
A4 The adversary chooses two cleartexts m_0 and m_1 and a label L. These are given to an "encryption oracle" that chooses $b \in \{0,1\}$ at random, encrypts m_b using the given label L, and gives the ciphertext c to the adversary.
A5 The adversary continues to interact with the uncorrupted servers, feeding them ciphertext/label pairs $(c', L') \neq (c, L)$.
A6 At the end of the game, the adversary outputs $b' \in \{0,1\}$.

Security against chosen ciphertext attack means that for any polynomial time bounded adversary $b' = b$ with probability only negligibly greater than $1/2$.

The above definition captures the notion of security against a *static* adversary (i.e., one who corrupts players only at the beginning of the protocol). All known practical threshold cryptosystems (including [21,17]) are provably secure only against such a static adversary; our protocol has the property as well.

4 Basic Tools

4.1 Threshold secret sharing

Let q be a prime, and $1 \leq k \leq n < q$. Shamir's [26] k out of n secret sharing scheme over $\mathbf{Z}_q$ works as follows. We have a random secret $x \in \mathbf{Z}_q$. We choose random points $f_1, \ldots, f_{k-1} \in \mathbf{Z}_q$, set $f_0 = x$, and define the polynomial $F(X) = \sum_{j=0}^{k-1} f_j X_j$. For $1 \leq i \leq n$, $x_i = F(i) \in \mathbf{Z}_q$ is the ith share of x. Just for notation purpose we will denote x as its own 0th share, $x = x_0 = f_0$.

If any subset of $k-1$ shares is revealed, then no information about x is obtained, whereas if k shares are revealed, x is completely determined, and can be computed by interpolation. Actually the following property holds: for any ordered subset $S = \{i_1, \ldots, i_k\} \subset \{0, \ldots, n\}$, and any $i \in \{0, \ldots, n\} \backslash S$, there exists an easy-to-compute sequence $\lambda_{i1}^S, \ldots, \lambda_{ik}^S \in \mathbf{Z}_q$, such that $x_i = \sum_{j=1}^{k} \lambda_{ij}^S x_{i_j}$.

4.2 Zero-knowledge proof of discrete logarithm identities

Let G be a group of prime order q with generators $g, \bar{g}$. Let $\mathsf{EDLog}_{g,\bar{g}}$ be the language of pairs $(u, \bar{u}) \in G^2$ such that $\log_g u = \log_{\bar{g}} \bar{u}$.

Our cryptosystems will heavily rely on a zero-knowledge proof *of membership* for the language $\mathsf{EDLog}_{g,\bar{g}}$. It is important to notice that our proofs techniques do

not require a proof of knowledge (which would create the problems encountered with the TDH0 cryptosystem).

The following is a well-known zero-knowledge proof system for $\mathsf{EDLog}_{g,\bar{g}}$, due to Chaum and Pedersen [7]. Although it happens to also be a proof of knowledge we will not use that property in our schemes.

Let $(u, \bar{u}) \in \mathsf{EDLog}_{g,\bar{g}}$ be given, so there exists $r \in \mathbf{Z}_q$ such that $u = g^r$ and $\bar{u} = \bar{g}^r$.

- The prover chooses $s \in \mathbf{Z}_q$ at random, computes $w = g^s$ and $\bar{w} = \bar{g}^s$, and sends $w, \bar{w}$ to the verifier.
- The verifier chooses $e \in \mathbf{Z}_q$ at random, sending this to the prover.
- The prover sends $f = s + re$ to the verifier. The verifier checks that $g^f = wu^e$ and $\bar{g}^f = \bar{w}\bar{u}^e$.

It is well known that this proof system is sound: the verifier can be cheated into accepting a pair not in the language with probability at most $1/q$. It is also well known that this proof system can be simulated in zero-knowledge against an *honest* verifier. By making the challenge e a hash of $(u, w, \bar{u}, \bar{w})$, then in the random hash function model, this becomes a non-interactive zero-knowledge proof.

Actually, a stronger soundness condition holds that we will need in the sequel: the argument that one uses to show soundness in the above non-interactive proof system can be used to show that when the verifier accepts, not only do we have (with overwhelming probability) $\log_g u = \log_{\bar{g}} \bar{u}$, but also $\log_g w = \log_{\bar{g}} \bar{w}$.

4.3 Intractability assumptions

Let G be a group with generator g.

The *Diffie-Hellman problem* is this: given g^x and g^y, compute g^{xy}, where x and y are random exponents.

The *Diffie-Hellman decision problem* is this: given a triple that is either of the form (g^x, g^y, g^{xy}) or (g^x, g^y, g^z), where x, y, and z are random exponents, determine which is the case.

Clearly, the second problem is no harder than the first, but it is not known if they are equivalent. The only known method for solving either problem is to solve the *discrete logarithm problem*: given g^x, compute x. For suitable groups, such as a large prime-order subgroup of the multiplicative group modulo a large prime, all of these problems are widely conjectured to be intractable.

5 The TDH1 Cryptosystem

We now describe the threshold cryptosystem TDH1.

TDH1 works over an arbitrary group G of prime order q, with generator g; for simplicity, assume that messages and labels are l-bit strings. It uses four hash functions:

$$H_1 : G \to \{0,1\}^l,\ H_2 : \{0,1\}^l \times \{0,1\}^l \times G \times G \to G,\ H_3, H_4 : G^3 \to \mathbf{Z}_q.$$

Key Generation. For a k out of n scheme, the key generation algorithm runs as follows (we assume $q > n$). Random points $f_0, \ldots, f_{k-1} \in \mathbf{Z}_q$ are chosen, defining a polynomial $F(X) = \sum_{j=0}^{k-1} f_j X^j \in \mathbf{Z}_q[X]$. For $0 \leq i \leq n$, set $x_i = F(i) \in \mathbf{Z}_q$ and $h_i = g^{x_i}$. For notational convenience, we set $x = F(0)$ and $h = h_0 = g^x$. The public key is $(h, h_1, \ldots, h_n)$, and the ith private key is x_i, for $1 \leq i \leq n$.

Note that the components $h_1, \ldots, h_n$ of the public key are only needed to verify decryption shares in the recovery algorithm, and are not needed by the encryption algorithm. In other words, the sender needs only to know h while the h_i should be made available to all the decryption servers.

The key generation protocol may be thought as being performed by a trusted dealer that self-destroys after the generation of the key. However using technique from [21] we may also assume that there is no need for a trusted dealer and that the players jointly generate the key pair.

Encryption. The algorithm to encrypt a message $m \in \{0,1\}^l$ with label $L \in \{0,1\}^l$ runs as follows. We choose $r, s \in \mathbf{Z}_q$ at random, and compute

$$c = H_1(h^r) \oplus m,\ u = g^r,\ w = g^s,\ \bar{g} = H_2(c, L, u, w),$$
$$\bar{u} = \bar{g}^r,\ \bar{w} = \bar{g}^s,\ e = H_3(\bar{g}, \bar{u}, \bar{w}),\ f = s + re.$$

The ciphertext is $(c, u, \bar{u}, e, f)$.

What is happening here is that the encryption includes a non-interactive proof that $\log_g u = \log_{\bar{g}} \bar{u}$.

Decryption. Decryption server i does the following given ciphertext $(c, u, \bar{u}, e, f)$ and label L. It checks if $e = H_3(\bar{g}, \bar{u}, \bar{w})$, where

$$w = g^f / u^e,\ \bar{g} = H_2(c, L, u, w),\ \bar{w} = \bar{g}^f / \bar{u}^e.$$

If this condition does not hold, it outputs "?". Otherwise, it proceeds as follows. It then chooses $s_i \in \mathbf{Z}_q$ at random, and computes

$$u_i = u^{x_i},\ u_i' = u^{s_i},\ h_i' = g^{s_i},\ e_i = H_4(u_i, u_i', h_i'),\ f_i = s_i + x_i e_i.$$

Its output is (u_i, e_i, f_i).

What is happening here is that the decryption share includes non-interactive proof that $\log_u u_i = \log_g h_i$. This is needed for robustness reason since it enforces the condition that the partial decryption is correct (see Theorem 1).

Recovery. The recovery algorithm takes as input a ciphertext $(c, u, \bar{u}, e, f)$, and a list of decryption shares. A share (u_i, e_i, f_i) is "good" if $e_i = H_4(u_i, u_i', h_i')$, where $u_i' = u^{f_i} / u_i^{e_i}$ and $h_i' = h^{f_i} / h_i^{e_i}$. Now assume we have a subset $S = \{i_1, \ldots, i_k\}$ of good shares. Then, using the notation defined in §4.1, the recovery algorithm outputs

$$m = H(\prod_{j=1}^{k} u_{i_j}^{\lambda_{0j}^S}) \oplus c.$$

Theorem 1. *In the random hash function model, the* TDH1 *cryptosystem is secure against chosen ciphertext attack, assuming the Diffie-Hellman problem in G is hard.*

The proof is in Appendix A. We also remark that it is straightforward to prove that in the random hash function model, the scheme is robust provided $n \geq 2k - 1$.

6 The TDH2 Cryptosystem

Cryptosystem TDH2 is very similar to TDH1. The main difference is that the group element $\bar{g}$, instead of changing with each encryption, is chosen at key-generation time.

We now give the details. As before, we have a group G of prime order q with generator g. We need three hash functions:

$$H_1 : G \rightarrow \{0,1\}^l, \; H_2 : \{0,1\}^l \times \{0,1\}^l \times G^4 \rightarrow \mathbf{Z}_q, \; H_4 : G^3 \rightarrow \mathbf{Z}_q.$$

Key Generation. Same as for TDH1, except that a random element $\bar{g} \in G$ is chosen which is also part of the public key.

Encryption. The algorithm to encrypt a message $m \in \{0,1\}^l$ with label $L \in \{0,1\}^l$ runs as follows. We choose $r, s \in \mathbf{Z}_q$ at random, and compute

$$c = H_1(h^r) \oplus m, \; u = g^r, \; w = g^s, \; \bar{u} = \bar{g}^r, \; \bar{w} = \bar{g}^s,$$
$$e = H_2(c, L, u, w, \bar{u}, \bar{w}), \; f = s + re.$$

The ciphertext is $(c, u, \bar{u}, e, f)$.

As in TDH1, the encryption includes a non-interactive proof that $\log_g u = \log_{\bar{g}} \bar{u}$.

Decryption. Decryption server i does the following given ciphertext $(c, u, \bar{u}, e, f)$ and label L. It checks that $e = H_2(c, L, u, w, \bar{u}, \bar{w})$, where $w = g^f / u^e$ and $\bar{w} = \bar{g}^f / \bar{u}^e$. If this condition does not hold, it outputs "?". Otherwise, it proceeds exactly as in TDH1, creating an output (u_i, e_i, f_i).

Recovery. Same as for TDH1.

Theorem 2. *In the random hash function model, the* TDH2 *cryptosystem is secure against chosen ciphertext attack, assuming the Diffie-Hellman* decision *problem in G is hard.*

The proof is in Appendix B. As for TDH1, the scheme is robust in the random hash function model if $n \geq 2k - 1$.

7 Implementation Issues

To implement these schemes, one has to choose concrete hash functions. This is relatively straightforward, but see [2] for a detailed discussion. One technicality that we have to deal with here, though, is the hash function H_1 in TDH1, whose output is supposed to be an element of the group G. For example, consider the case where p is a prime, $p-1 = mq$, $(m, q) = 1$, and G is the group of order q in $\mathbf{Z}_p^*$. We could implement H_1 by raising the output of a standard hash function (viewed as a number) to the power m modulo p. This gives us an element in G. Note that the decryption and recovery algorithms must also check that the given group elements lie in G. It is straightforward to modify the proof of security to deal with this.

Unfortunately, this implementation of H_1 is quite costly, as it requires extra exponentiations, some to the power m, which is typically much larger than q.

The TDH2 scheme does not suffer from this problem. Moreover, in TDH2, the group element $\bar{g}$ is fixed (per public key). In practice, this makes quite a difference, as one can pre-compute a table that makes exponentiation to the base $\bar{g}$ far more efficient than when it is constantly changing [6,19]. This speeds up the encryption algorithm significantly. Of course the same can be done for g already in TDH1.

8 Conclusion

We have proposed two new threshold cryptosystems, TDH1 and TDH2, that are provably secure in the random hash function model assuming, respectively, that the Diffie-Hellman and Diffie-Hellman decision problems are hard.

TDH2 requires a stronger intractability assumption than TDH1, but is much more efficient. Moreover, TDH2 is not much less efficient than the very simple TDH0 scheme in §2.4, which is not known to be secure in the random hash function model.

We close with three open problems: (1) determine the security of TDH0; (2) find a practical threshold cryptosystem based on RSA that is provably secure in the random hash function model; (3) find practical and provably secure threshold cryptosystems, without the random hash function model.

Appendix A: Proof of Theorem 1

We show how to use an adversary that can guess bit b in game A to solve the Diffie-Hellman problem. It is clear that if the adversary is to guess bit b, then he must query the function H_1 at the same point that the encryption oracle did. We simulate the adversary's view up to the point that this happens. After this point, the simulation is no longer perfect, but it does not matter: we already solved the Diffie-Hellman problem. Actually, the output of our algorithm is simply a list of all points at which H_1 was queried, which with nonnegligible probability will

contain the solution to the Diffie-Hellman problem. Techniques in [27] can be used to transform this into an algorithm that outputs a single, correct solution to the Diffie-Hellman problem.

We now give the details of the simulation. Let $\alpha, \beta \in G$ random elements in G for which we want to solve the Diffie-Hellman problem to the base g. That is, we want to compute $\gamma = \alpha^{\log_g \beta}$.

At any point in the simulation, the adversary may query one of the random hash functions. The simulator responds by first checking if the value of the hash function has already been defined at the given point; if so, it responds with the defined value; otherwise, it chooses a random value, defines the value of the hash function at the given point to be this value, and responds with this value.

The simulator itself may at some point choose to define the value of a hash function at a chosen point. Such "backpatching" is allowable so long as the hash function has not already been defined at the chosen point.

Now suppose the adversary in step A1 chooses to corrupt a subset of $k-1$ servers. Without loss of generality, we can assume these are servers $P_1, \ldots, P_{k-1}$. Let $S = \{0, \ldots, k-1\}$, and we will write λ_{ij} instead of λ_{ij}^S.

Now in step A2, we proceed as follows. We choose $x_1, \ldots, x_{k-1} \in \mathbf{Z}_q$ at random, and we set $h = \alpha$. Then for $k \leq i \leq n$, we compute $h_i = h^{\lambda_{i0}} \prod_{j=1}^{k-1} g^{x_j \lambda_{ij}}$.

Next, we have to describe how to simulate the "encryption oracle" in step A4, and how to simulate each query to one of the noncorrupt decryption servers.

We deal first with the encryption oracle. The adversary gives a label $\mathbf{L}$ and two messages, m_0 and m_1, to the encryption oracle. We ignore the messages completely. Instead, we simply choose $\mathbf{c} \in \{0,1\}^l$ and $\mathbf{t}, \mathbf{e}, \mathbf{f} \in \mathbf{Z}_q$ at random. We then set

$$\mathbf{u} = \beta,\ \bar{\mathbf{g}} = g^{\mathbf{t}},\ \bar{\mathbf{u}} = \mathbf{u}^{\mathbf{t}},\ \mathbf{w} = g^{\mathbf{f}}/\mathbf{u}^{\mathbf{e}},\ \bar{\mathbf{w}} = \bar{\mathbf{g}}^{\mathbf{f}}/\bar{\mathbf{u}}^{\mathbf{e}}.$$

We then backpatch, defining $H_2(\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w}) = \bar{\mathbf{g}}$, and $H_3(\bar{\mathbf{g}}, \bar{\mathbf{u}}, \bar{\mathbf{w}}) = \mathbf{e}$. The output of the encryption oracle is $(\mathbf{c}, \mathbf{u}, \bar{\mathbf{u}}, \mathbf{e}, \mathbf{f})$.

It is easily verified that this backpatching is allowable. Also, one sees that $\mathbf{u}$, $\bar{\mathbf{g}}$, and $\bar{\mathbf{u}}$ have the right distribution; namely, they are random, subject to to the condition $\log_g \mathbf{u} = \log_{\bar{\mathbf{g}}} \bar{\mathbf{u}}$. The rest is just a standard zero-knowledge simulation. Thus, simulation is perfect, and will remain perfect, as long as the adversary does not query H_1 at the point $\gamma = \mathbf{u}^{\log_g h}$. Note that the simulator never directly queries or backpatches H_1 itself; it only does this upon request of the adversary.

We next deal with the simulation of the uncorrupted decryption servers. First of all, whenever the adversary queries H_2 at a point other than $(\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w})$, we arrange that the simulator defines the value $\bar{g}$ at that point by first choosing $t \in \mathbf{Z}_q$ at random, and then computing $\bar{g} = h^t$, so that the simulator knows $\log_h \bar{g}$ (but the adversary is oblivious to this).

Now suppose P_i is given a valid ciphertext/label pair $((c, u, \bar{u}, e, f), L) \neq ((\mathbf{c}, \mathbf{u}, \bar{\mathbf{u}}, \mathbf{e}, \mathbf{f}), \mathbf{L})$. Now, $((c, u, \bar{u}, e, f), L)$ determines via the validity conditions corresponding variables $\bar{g}$, w, $\bar{w}$.

We first argue that we can assume that $(c, L, u, w) \neq (\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w})$. On the contrary, suppose that $(c, L, u, w) = (\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w})$. Then of course $\bar{g} = \bar{\mathbf{g}}$. But then

with overwhelming probability, we must also have $(\bar{u}, \bar{w}) = (\bar{\mathbf{u}}, \bar{\mathbf{w}})$. This easily follows from the strong soundness condition discussed in §4.2. It then follows that $e = \mathbf{e}$ and $f = \mathbf{f}$, which contradicts our assumption that $((c, u, \bar{u}, e, f), L) \neq ((\mathbf{c}, \mathbf{u}, \bar{\mathbf{u}}, \mathbf{e}, \mathbf{f}), \mathbf{L})$.

So assume $(c, L, u, w) \neq (\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w})$. We can assume that the adversary has already queried H_2 at the point (c, L, u, w), so that we have $\bar{g} = H_2(c, L, u, w) = h^t$, where t is known to the simulator, as discussed above.

Now suppose $u = g^r$, where r is not known to the simulator. We want to compute h^r. But by the soundness of the proof that $\log_g u = \log_{\bar{g}} \bar{u}$, we can assume that $\bar{u} = \bar{g}^r$. But then $(\bar{u})^{1/t} = (\bar{g})^{r/t} = h^r$.

So the simulator can compute h^r, but we are not quite done. We want to simulate the output of server P_i, who is supposed to output $u_i = h_i^r$, along with a proof that $\log_u u_i = \log_g h_i$. But u_i can be computed by the simulator as $u_i = (\bar{u})^{\lambda_{i,0}/t} \prod_{j=1}^{k-1} u^{x_j \lambda_{i,j}}$. Once we have u_i, we can readily produce a zero-knowledge simulation of the proof that $\log_u u_i = \log_g h_i$, backpatching H_4 as necessary.

That completes the proof of Theorem 1.

Appendix B: Proof of Theorem 2

Again, the proof is by reduction, and we assume the adversary queries, with nonnegligible probability, the same point in game A that was queried by the encryption oracle in step A4.

Let (α, β, γ) be a random instance of the Diffie-Hellman decision problem. This triple is drawn from one of two distributions: that of *Diffie-Hellman triples*, where $\alpha = g^x$, $\beta = g^y$, and $\gamma = g^{xy}$, for random $x, y \in \mathbf{Z}_q$, or from that of *random triples*, where $\alpha = g^x$, $\beta = g^y$, and $\gamma = g^z$, for random $x, y, z \in \mathbf{Z}_q$. The job of the simulator is to distinguish between these two distributions. It outputs a 1 or a 0, and to be an effective test, the expected value of its output on the two distributions should differ by a nonnegligible amount.

We simulate the view of the adversary in game A as follows.

As in the proof of Theorem 1, we assume the adversary corrupts players $P_1, \ldots, P_{k-1}$ in step A1. In step A2, we set $h = \alpha (= g^x)$, generate $x_1, \ldots, x_{k-1} \in \mathbf{Z}_q$ at random, and solve for $h_k, \ldots, h_n$ as in the proof of Theorem 1. We also choose $t \in \mathbf{Z}_q$ at random and set $\bar{g} = h^t (= g^{xt})$.

Now we discuss how to simulate the adversary's view of the encryption oracle in step A4, given a label $\mathbf{L}$. We choose $\mathbf{c} \in \{0,1\}^l$ at random. We set $\mathbf{u} = \beta (= g^y)$ and $\bar{\mathbf{u}} = \gamma^t$, which is either g^{xyt} or g^{zt}, depending on the distribution from which (α, β, γ) was drawn. We then choose $\mathbf{e}, \mathbf{f} \in \mathbf{Z}_q$ at random, and compute $\mathbf{w} = g^{\mathbf{f}}/\mathbf{u}^{\mathbf{e}}$, and $\bar{\mathbf{w}} = \bar{g}^{\mathbf{f}}/\bar{\mathbf{u}}^{\mathbf{e}}$. We then backpatch, setting $H_2(\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w}, \bar{\mathbf{u}}, \bar{\mathbf{w}}) = \mathbf{e}$. The output of the encryption oracle is $(\mathbf{c}, \mathbf{u}, \bar{\mathbf{u}}, \mathbf{e}, \mathbf{f})$.

The simulation of the uncorrupted servers is essentially just as it was in the proof of Theorem 1: the key is that the simulator knows t with $\bar{g} = h^t$, and so given a valid ciphertext/label pair $((c, u, \bar{u}, e, f), L) \neq ((\mathbf{c}, \mathbf{u}, \bar{\mathbf{u}}, \mathbf{e}, \mathbf{f}), \mathbf{L})$, it is

easy to argue that with overwhelming probability $\log_g u = \log_{\bar{g}} \bar{u}$, which implies we can compute u^x as $(\bar{u})^{1/t}$, and simulate the rest of the server's output just as before.

The simulator itself never directly queries or backpatches H_1, except on behalf of the adversary. If the adversary ever queries H_1 at γ, we stop and output 1; otherwise, if the adversary terminates without querying H_1 at γ, we output 0.

That completes the description of the simulator.

Consider the joint distribution of $(h, \bar{g}, \mathbf{u}, \bar{\mathbf{u}})$. In the case where (α, β, γ) drawn from the Diffie-Hellman triple distribution, $(h, \bar{g}, \mathbf{u}, \bar{\mathbf{u}})$ is (statistically indistinguishable from) a random element of G^4, subject to the condition $\log_g \mathbf{u} = \log_{\bar{g}} \bar{\mathbf{u}}$. In the case where (α, β, γ) is a random triple, $(h, \bar{g}, \mathbf{u}, \bar{\mathbf{u}})$ is simply (statistically indistinguishable from) a random element of G^4. In either case, γ is determined by $\gamma = \bar{\mathbf{u}}^{\log_{\bar{g}} h}$; moreover, if (α, β, γ) is a Diffie-Hellman triple, then the relation $\gamma = \mathbf{u}^{\log_g h}$ also holds.

We now argue as follows. In the case where (α, β, γ) is drawn from the Diffie-Hellman triple distribution, the simulation of game A is perfect until the adversary queries H_1 at $\gamma = \mathbf{u}^{\log_g h}$, at which point we stop and output a 1. By our assumption about the behavior of the adversary, and the fact that the simulation is perfect up to this point, this happens with nonnegligible probability.

Now, if in the case where (α, β, γ) is a random triple the simulator outputs a 1 with negligible probability, we are done: the simulator is an effective test for distinguishing Diffie-Hellman triples from random triples.

Otherwise, suppose that in the case where (α, β, γ) is a random triple the simulator outputs 1 with nonnegligible probability. As mentioned above, $(h, \bar{g}, \mathbf{u}, \bar{\mathbf{u}})$ is just a random element in G^4. The other random variables $\mathbf{e}$, $\mathbf{f}$, $\mathbf{w}$, and $\bar{\mathbf{w}}$ are also just random, subject to relations that make the "proof" that $\log_g \mathbf{u} = \log_{\bar{g}} \bar{\mathbf{u}}$ look legitimate; in fact, the relation $\log_g \mathbf{u} = \log_{\bar{g}} \bar{\mathbf{u}}$ does not in general hold, and the "proof" is entirely bogus, but that is irrelevant.

The point is that if the adversary makes the simulator output a 1, it can essentially compute $\bar{\mathbf{u}}^{\log_{\bar{g}} h}$ given random $(h, \bar{g}, \mathbf{u}, \bar{\mathbf{u}}) \in G^4$. As we show below, we can use this adversary to solve the following variant of the Diffie-Hellman problem with the nonnegligible probability: given random $\alpha' = g^t$ and $\beta' = g^v$, compute $\gamma' = g^{v/t}$. It is easy to see that this problem is equivalent (under polynomial-time reduction) to the Diffie-Hellman problem, and is certainly at least as hard as the Diffie-Hellman decision problem.

Now the details. The new simulation proceeds as follows. The input to the simulator is α', β' as above. First choose $x \in \mathbf{Z}_q$ at random, set $\bar{g} = (\alpha')^x (= g^{xt})$, and run the actual key generation algorithm for the cryptosystem, in particular, setting $h = g^x$. Since this new simulator knows the private decryption key, it can without any trouble respond to arbitrary decryption requests.

Now consider the encryption oracle in step A4, given label $\mathbf{L}$. We choose $\mathbf{c} \in \{0, 1\}$ at random, $\mathbf{e}, \mathbf{f} \in \mathbf{Z}_q$ at random, and $\mathbf{u} \in G$ at random. We then set $\bar{\mathbf{u}} = \beta' (= g^v)$. We compute $\mathbf{w} = g^{\mathbf{f}} / \mathbf{u}^{\mathbf{e}}$, and $\bar{\mathbf{w}} = \bar{g}^{\mathbf{f}} / \bar{\mathbf{u}}^{\mathbf{e}}$. We then backpatch, setting $H_2(\mathbf{c}, \mathbf{L}, \mathbf{u}, \mathbf{w}, \bar{\mathbf{u}}, \bar{\mathbf{w}}) = \mathbf{e}$. The output of the encryption oracle is $(\mathbf{c}, \mathbf{u}, \bar{\mathbf{u}}, \mathbf{e}, \mathbf{f})$.

This new simulator halts when the adversary halts, outputting the list of all queries made to H_1.

It is straightforward to verify that the view of this adversary relative to this new simulator is identical to the view of the adversary relative to the original simulator on a random triple, at least up until the point that it queries H_1 at

$$\gamma = \bar{u}^{\log_g h} = (g^v)^{x/xt} = g^{v/t} = \gamma'.$$

So, if the adversary causes the first simulator on a random triple to output 1 with nonnegligible probability, then this same adversary causes this new simulator to output a list containing the desired solution γ'.

That completes the proof of Theorem 2.

References

1. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. Preprint, 1997.
2. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, 1993.
3. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Crypto '94*, pages 92–111, 1994.
4. M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive zero knowledge. *SIAM J. Comput.*, 6(4):1084–1118, 1991.
5. C. Boyd. Digital multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Claredon Press, 1986.
6. E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology–Eurocrypt '92*, pages 200–207, 1992.
7. D. Chaum and T. Pederson. Wallet databases with observers. In *Advances in Cryptology–Crypto '92*, pages 89–105, 1992.
8. I. Damgard. Towards practical public key cryptosystems secure against chosen ciphertext attacks. In *Advances in Cryptology–Crypto '91*, pages 445–456, 1991.
9. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *26th Annual ACM Symposium on Theory of Computing*, pages 522–533, 1994.
10. A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *33rd Annual Symposium on Foundations of Computer Science*, 1992.
11. Y. Desmedt. Society and group oriented cryptography: a new concept. In *Advances in Cryptology–Crypto '87*, pages 120–127, 1987.
12. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology–Crypto '89*, pages 307–315, 1989.
13. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22:644–654, 1976.
14. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.
15. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology—Crypto '86*, pages 186–194, 1986.

16. Y. Frankel and M. Yung. Cryptanalysis of immunized LL public key systems. In *Advances in Cryptology–Crypto '95*, pages 287–296, 1995.
17. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Advances in Cryptology–Crypto '96*, pages 157–172, 1996.
18. C. H. Lim and P. J. Lee. Another method for attaining security against adaptively chosen ciphertext attacks. In *Advances in Cryptology–Crypto '93*, pages 420–434, 1993.
19. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology–Crypto '94*, pages 95–107, 1994.
20. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.
21. T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology–Eurocrypt '91*, pages 522–526, 1991.
22. D. Pointcheval and J. Stern. Provably secure blind signature schemes. In *Advances in Cryptology–Asiacrypt '96*, pages 252–265, 1996.
23. 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.
24. 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.
25. C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.
26. A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
27. V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology–Eurocrypt '97*, 1997.
28. Y. Zheng and J. Seberry. Practical approaches to attaining security against adaptively chosen ciphertext attacks. In *Advances in Cryptology–Crypto '92*, pages 292–304, 1992.