J. Cryptology (2002) 15: 75–96
DOI: 10.1007/s00145-001-0020-9

Journal of
CRYPTOLOGY

Securing Threshold Cryptosystems against Chosen Ciphertext Attack[*]

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

Rosario Gennaro
IBM T. J. Watson Research Center,
P.O.Box 704, Yorktown Heights, NY 10598, U.S.A.
rosario@watson.ibm.com

Communicated by Ueli Maurer

Received November 2000 and revised September 2001
Online publication 11 March 2002

Abstract. For the most compelling applications of threshold cryptosystems, security against chosen cipher text attack is a requirement. However, prior to the results presented here, there appeared to be no practical threshold cryptosystems in the literature that were provably chosen ciphertext secure, even in the idealized random oracle 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 oracle model. Not only are these protocols computationally very efficient, but they are also non-interactive, which means they can be easily run over an asynchronous communication network.

Key words. Public key encryption, Threshold cryptography, Chosen ciphertext attack.

1. Introduction

In a threshold cryptosystem the secret key of a public key cryptosystem is shared among a set of decryption servers, so that a quorum of servers can act together to decrypt a given ciphertext. Just as for ordinary, non-threshold cryptosystems, a natural and very useful notion of security is that of security against chosen ciphertext attack. 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.

[*] A preliminary version of this paper appears in the *Proceedings of EuroCrypt* '98

Even though the most compelling applications of threshold cryptosystems seem to require chosen ciphertext security, prior to the results presented here, there appeared to be no practical threshold cryptosystems in the literature that were provably secure—even in the random oracle model, where one models a cryptographic hash function as a random oracle.

Our main contribution is to present and analyze two such schemes which are secure in the random oracle model. The first scheme, which we call `TDH1` (for Threshold Diffie–Hellman), is secure assuming the hardness of the *computational* Diffie–Hellman problem [DH]. The second scheme, `TDH2`, is secure under the stronger assumption of the hardness of the *decisional* Diffie–Hellman problem, but is more efficient than `TDH1`.

2. Background and Related Work

2.1. *Chosen Ciphertext Security*

In the context of ordinary, non-threshold cryptosystems, the notion of security against chosen ciphertext attack was developed by Naor and Yung [NY], Rackoff and Simon [RS], and Dolev et al. [DDN]. These definitions are further explored and developed in [BDPR] and [BS].

In a chosen ciphertext attack, the adversary is given access to a *decryption oracle* that allows him to obtain the decryptions of ciphertexts of his choosing. Intuitively, security in this setting means that an adversary obtains (effectively) no information about encrypted messages, provided the corresponding ciphertexts are never submitted to the decryption oracle.

2.2. *Threshold Cryptosystems*

In a k out of n threshold cryptosystem there is a single public encryption key, but the corresponding private decryption key is shared among a set of n decryption servers in such a way that k of them must cooperate to decrypt a message.

We consider only simple client/server protocols. That is, to decrypt a message, a client presents a ciphertext to be decrypted to a server, who responds with a *decryption share*. The client should be able to inspect the decryption share and verify its correctness or "validity." After collecting valid shares from k servers, the client can combine these shares to obtain the decryption of the ciphertext.

Such client/server protocols are attractive since they require no interaction or synchronization among the servers, and as such can be easily and efficiently run on an asynchronous communications network, with absolutely no reliance on network latency guarantees. In practice, this is an important property, since it allows the servers to be geographically distributed, and it allows the use of inexpensive public communication networks, rather than the expensive, private networks that would be required to provide a guaranteed latency.

For a k out of n threshold cryptosystem, the adversary first corrupts $k - 1$ decryption servers. After corrupting these servers, the key generation algorithm is run, and the adversary obtains the shares of the secret key held by the corrupted servers.

During the course of the attack, the adversary submits ciphertexts to the uncorrupted decryption servers; whenever the adversary submits a ciphertext ψ to a server, the server responds with its decryption share of ψ.

Intuitively, security in this setting means that the adversary obtains (effectively) no information about encrypted messages, provided their corresponding ciphertexts are never submitted to an uncorrupted server.

Note that we only analyze the situation where the adversary *statically* corrupts the servers, i.e., it makes its decision as to who to corrupt independently of the observed network traffic.

One important difference between chosen ciphertext attack in the non-threshold and threshold settings is that in the latter the adversary sees not only the decryption of chosen ciphertexts, but also the decryption shares of these ciphertexts. This extra information that is available to the adversary can make security proofs more challenging.

2.3. *Applications of Threshold Cryptosystems*

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 or the cost of creating a ciphertext compared with 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 *decryption 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 and its current state. One can 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 securely 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.

There are other similar scenarios where a secret of some sort needs to be "escrowed" by encrypting it under a trusted third party's public key, and where this third party only decrypts ciphertexts according to a particular decryption policy. One such example is the recent work of [ASW] on *fair exchange*, where an "off line" trusted third party is used to enforce fairness.

Another application of threshold decryption is to maintain causal order among client requests for a distributed or replicated service [RB], [CKPS]. In this scenario, clients make requests to a distributed service. One wants each server to process the same requests in the same order, and in addition, one wants to prevent an adversary from inserting requests that depend on other requests that have been issued but not yet processed. This can be achieved using a threshold cryptosystem that is secure against chosen ciphertext attack, as discussed rather informally in [RB] and more rigorously in [CKPS].

Yet another application of threshold decryption is *verifiable signature sharing* [FR], [CG2]. In a signature sharing scheme, one party wants to distribute verifiably shares of a digital signature to a group of players so that later, a quorum of these players can combine their shares to reconstruct the digital signature. One way to implement this is to combine an algorithm to encrypt verifiably a signature under a public key, and have the corresponding decryption key distributed using a threshold cryptosystem. The verifiable encryption can be done using the algorithms in [ASW]. The use of a label can also be helpful here to control the circumstances under which the digital signature will be reconstructed.

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 [LL1] (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 would have to see the decrypted labeled plaintext before it is output, thus it would be too late at that point to check if the label is correct, since the attacker has already seen the result of the decryption operation.

2.4. *Constructions of Chosen Ciphertext Secure Cryptosystems*

A number of ordinary, non-threshold chosen ciphertext secure cryptosystems have been proposed in the literature.

In addition to formal definitions, [NY], [RS], and [DDN] present provably secure cryptosystems (without random oracles). However, all of these schemes rely on theoretical constructions of non-interactive zero-knowledge proofs [BDMP], and as such are quite impractical.

To overcome the above inefficiency problem some practical cryptosystems intended to be secure against chosen ciphertext attack were proposed by Damgård [Da], Zheng and Seberry [ZS], and Bellare and Rogaway [BR1], [BR2]. The scheme in [BR1] was proven chosen ciphertext secure in the random oracle model, using any one-way trapdoor permutation, such as RSA [RSA]. The scheme in [BR2], known as OAEP, was also claimed to be chosen ciphertext secure, using any one-way trapdoor permutation; however, it was shown in [Sh5] that the proof was invalid and could not be repaired using standard techniques, at least for an arbitrary one-way trapdoor permutation. It was also shown in [Sh5] that OAEP when instantiated with low-exponent RSA was in fact chosen ciphertext secure, and this result was extended to arbitrary-exponent RSA in [FOPS].

Recently, the first truly practical cryptosystem that is provably secure against chosen ciphertext attack *without* using random oracles was discovered by Cramer and Shoup [CS]. The security of this scheme is based on the hardness of the decisional Diffie–Hellman problem. For the historical record, we should point out that the results in [CS] follow and build on the results in a preliminary version of this paper [SG]. The paper [Sh4] presents a variant of the scheme in [CS] that is both more practical and (potentially) more secure. Subsequent to [CS] and [SG], Fujisaki and Okamoto [FO] presented a cryptosystem that can be proven secure against chosen ciphertext attack in the random oracle model under the computational Diffie–Hellman assumption; this result was refined and extended in [Po] and [BLK].

Although the schemes in this paper are presented as threshold cryptosystems, they can also be used as ordinary cryptosystems, and as such are the first cryptosystems in the literature based on the Diffie–Hellman problem that are chosen ciphertext secure in the random oracle model. The subsequent schemes of [FO] and [Po] are more efficient; however, they cannot be readily transformed into threshold cryptosystems. The papers [TY], [ABR1], and [ABR2] also present schemes based on the Diffie–Hellman problem, but to date, they are not known to be chosen ciphertext secure, even in the random oracle model (at least, assuming a standard intractability assumption).

2.5. *Difficulties in Securing Threshold Cryptosystems against Chosen Ciphertext Attack*

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

It should be observed that none of the *practical* non-threshold schemes mentioned above can be readily transformed into threshold schemes that are chosen ciphertext secure. To see why, consider the scheme in [BR1], 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 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 for the function f. 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. Of course, one could use general techniques for multi-party computation, but this would be extremely impractical.

The above difficulty was noted by Lim and Lee [LL1], 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 [FY].

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. It is essential that this proof of knowledge allows the simulator to extract the plaintext "on line," without any "rewinding." 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.

Since the publication of the Cramer–Shoup cryptosystem [CS], several threshold implementations of it have been designed and proved secure (also without random oracles) [CG1], [Ab], [JL]. The validity test of the Cramer–Shoup cryptosystem is *not* publicly checkable, which, as we have seen, makes it difficult to distribute the decryption function efficiently. Nevertheless, the above results show how the special algebraic structure of the validity test can be exploited to obtain a protocol that is much more efficient than a general multi-party computation. We should stress, however, that none of the schemes in [CG1], [Ab], and [JL] are nearly as practical as the schemes we present in this paper. In particular, none of these schemes is a simple, non-interactive, client/server protocol like ours. All of them require either a large degree of synchronized interaction or storage for a large number of pre-shared secrets (proportional to the total number of decryption requests that may be performed over the lifetime of the system). It remains an open problem to construct a truly practical, non-interactive, client/server threshold cryptosystem secure against chosen ciphertext attack, whose security proof does not rely on the random oracle model.

2.6. *The Random Oracle Model*

The random oracle model was first used by Fiat and Shamir [FS], and later given a more rigorous treatment by Bellare and Rogaway [BR1]. It has proved to be quite useful in analyzing a wide range of cryptographic schemes and protocols. See, for example, [BR1]–[BR3], [BMP], [CKS], [PS2], and [Sh3].

The random oracle methodology works as follows. Consider a cryptographic scheme that makes use of cryptographic hash functions (like SHA-1 or MD5). Instead of analyzing the security of this scheme directly, we analyze its security in an *idealized* model of computation where the hash functions are replaced by "black boxes" that output random strings. More specifically, all parties involved, including the adversary, do not evaluate the hash function directly, but only have oracle access to this function; moreover, the function implemented by this oracle is a *random* function: whenever the oracle is queried at a new input, it outputs a random bit string independent of all other oracle outputs. Note that the oracle implements a function, in the sense that if it is given the same input twice, the two outputs are the same.

The basic tenet of the random oracle model approach is to view a proof of security in the random oracle model as "strong evidence" that the scheme is actually secure in the standard model, i.e., the "real world."

When one instantiates the random oracle with an actual hash function, it is important to apply the following "rule of thumb": the actual hash function should be in some vague sense "independent" of other computations performed by the algorithms in the scheme; i.e., there should be no "obvious" interactions or correlations between the hash function and other computations.

When the random oracle is instantiated with a hash function, this hash function (i.e., its description as an algorithm) becomes known to the adversary, and so a real world adversary could of course exploit special properties of the hash function, which is something that he could not do with a random oracle. Thus, a proof of security in the random oracle model at most implies security in the real world against adversaries who never look at the description of the hash function, and only access it as a black box, as in the random oracle model.

The limitations of the random oracle model were demonstrated by Canetti et al. [CGH]. They exhibit cryptographic schemes that are secure in the random oracle model, but are trivially insecure in the real world with *any* instantiation of the random oracle. The schemes they exhibit are quite unnatural, and their results in no way uncover weaknesses in any protocols in the literature that have been proven secure in the random oracle model. One lesson to be drawn from their work is that the above-mentioned "rule of thumb" cannot be formulated as a simple syntactic constraint, and that the application of this "rule" is destined to remain an "art" rather than a "science."

Despite these limitations, the random oracle model seems to be a very good heuristic. All things being equal, a proof of security in the real world is to be preferred; however, if substantially more efficient schemes can be designed that can only be analyzed in the random oracle model, then these schemes deserve a place in the security engineer's toolbox. Certainly, a proof of security in the random oracle model is far better than no proof of security at all.

2.7. *The* `TDH0` *Cryptosystem*

In this section we discuss a simple threshold cryptosystem, based on the computational Diffie–Hellman assumption, which we call `TDH0`. This cryptosystem (in both threshold and non-threshold form) has been claimed in several papers to be chosen ciphertext secure; however, we argue that these claims are unjustified—even in the random oracle model.

Say we have a group G of prime order q with generator g, hash functions H and H', and a public key $h = g^x$. To encrypt a message m with label L, 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 [Sc] signature scheme with "public key" u and "private key" r. More specifically, to generate this proof of knowledge, we compute $w = g^s$ for random $s \in \mathbf{Z}_q$, $e = H'(c, L, u, w)$, and $f = s + re \in \mathbf{Z}_q$. The proof of knowledge consists of the pair (e, f). The entire ciphertext is then $\psi = (c, L, u, e, f)$. To verify the validity of the ciphertext, one checks that $e = H'(c, L, u, w)$, where $w = g^f/u^e$.

Intuitively, this strategy makes sense, since if the adversary proves that he "knows" the decryption of a ciphertext, then giving the adversary this decryption should not help him. The trouble is, this intuition cannot be transformed into a formal proof of security. TDH0 may very well be secure, but it does not seem possible to prove, using known techniques, a reduction to any standard cryptographic assumption, *even in the random oracle model*.

The problem is a bit subtle. Schnorr's interactive identification scheme (from which the signature scheme is derived using the Fiat–Shamir heuristic) is a proof of knowledge (see [FFS] and [BG] for definitions). However, the corresponding knowledge extractor does not operate "on line"—it must "rewind" the adversary. More specifically, in our setting, if the adversary requests the decryption of $\psi = (c, L, u, e, f)$, we have to rewind the adversary back to the point where it queried the random oracle H' with input $(c, L, u, g^f/u^e)$, and feed the adversary a different challenge e'. Then if we run the adversary forward, we hope that he makes a valid decryption request $\psi' = (c, L, u, e', f')$, with f' such that $g^f/u^e = g^{f'}/u^{e'}$. If and when he does this, we can compute $r = \log_g u$ as $r = (f - f')/(e - e')$. Once we have r, we can decrypt the original ψ.

Unfortunately, when one tries to turn the above idea into a proof, one discovers that the running time of the simulator can blow up exponentially. A similar phenomenon was observed by Pointcheval and Stern [PS1] in the context of blind digital signatures.

We illustrate the problem with an example. In this example, we just consider TDH0 as an ordinary, non-threshold scheme, since this is simpler, and is sufficient to illustrate the point. Our adversary works as follows. He generates a sequence of ciphertexts $\psi_1, \ldots, \psi_t$, of the form $\psi_i = (c_i, L_i, u_i, e_i, f_i)$, for $1 \le i \le t$. The adversary generates these ciphertexts in the usual way, except that to generate $u_i = g^{r_i}$, he computes r_i as some sort of hash of $e_1, \ldots, e_{i-1}$. Likewise, the message m_i that he encrypts is also a hash of $e_1, \ldots, e_{i-1}$. The adversary generates these ciphertexts in this order, accessing the random oracle for H' to generate $e_1, \ldots, e_t$. Next, he proceeds to obtain decryptions—*in reverse order*—of $\psi_t, \ldots, \psi_1$. To summarize, the adversary accesses the random oracle t times, obtaining $e_1, \ldots, e_t$, and then accesses the decryption oracle t times, submitting $\psi_t, \ldots, \psi_1$ for decryption.

Now, imagine how a simulator would work. We want to simulate the responses to the adversary's random oracle and decryption oracle requests, without knowing the secret key x of the cryptosystem. When the adversary makes his first decryption request, ψ_t, we have to rewind the adversary to the point where he obtained the challenge e_t from the random oracle H', and feed the adversary a different challenge e'_t. Then we run him forward, and when he presents a corresponding ψ'_t for decryption, we extract r_t, and so we can easily decrypt ψ_t. Now we let the adversary move forward to the second decryption request, ψ_{t-1}. As before, we rewind the adversary to the point where he obtained the challenge e_{t-1} from from the random oracle H', and feed the adversary a different challenge e'_{t-1}. When we run him forward, before the adversary makes his request to decrypt the corresponding ψ'_{t-1}, he inconveniently asks us first to decrypt a ciphertext $\psi''_t = (c''_t, L''_t, u''_t, e''_t, f''_t)$. Unfortunately, we cannot directly respond to this request, since (in general) $r''_t \neq r_t$ and $m''_t \neq m_t$; this is because both r''_t and m''_t are computed as a hash of $e_1, \ldots, e_{t-2}, e'_{t-1}$ and $e'_{t-1} \neq e_{t-1}$. So we will recursively have to rewind the adversary back to the point where he obtained e''_t from the random oracle, and run him forward again, just so that we can respond to the decryption request for ψ''_t.

It should now be clear that this adversary will force the simulator to run for time proportional to 2^t. We do not claim that TDH0 is insecure, but any proof of security will have to circumvent this exponential blow up in the simulation. Because of this difficulty, it appears that current proof techniques are not adequate to prove the security of TDH0.

One could circumvent all this by straightaway assuming an on-line knowledge extractor; that is, we simply *assume* that any algorithm that can create a valid proof of knowledge can be transformed into an algorithm that simultaneously outputs a corresponding witness. Such an assumption is made by Tsiounis and Yung [TY] in the analysis of a non-threshold version of TDH0. A similar type of assumption is made by Damgård [Da] and Zheng and Seberry [ZS]. This type of assumption, however, is not very acceptable: it is completely non-standard, and it is not an "intractability assumption" in the usual sense of the term.

Other papers ([Ja], [DK], as well as a preliminary version of [TY]) have claimed that TDH0 is secure, without offering any proof at all beyond a vague argument that "since the Schnorr signature scheme is a proof of knowledge, accessing the decryption oracle does not help." As we have seen, the notion of a "proof of knowledge" is not always that useful, especially when the "knowledge extractor" requires rewinding.

3. A Formal Security Model

In this section we present a formal model for a k out of n threshold cryptosystem. Before giving the details, we briefly sketch the overall workings of such a system.

For simplicity, in the following we assume that the system is initialized by a trusted dealer that gives the decryption servers a share of the private key. It is important to notice though that this trusted dealer can be replaced by a secure communication protocol among the servers at the end of which a public key is generated and the servers have shares of the matching decryption key. For the specific case of our threshold cryptosystems TDH1 and TDH2, we can use the key generation protocol of [GJKR].

Operation of the cryptosystem runs as follows.

There is a trusted dealer and a set $P_1, \ldots, P_n$ of decryption servers.

In an *initialization phase*, the dealer is run, creating a public key PK, a verification key VK, and private keys $\vec{SK} = (SK_1, \ldots, SK_n)$. For $1 \leq i \leq n$, the private key SK_i is given to server P_i.

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

A user who wants to decrypt a ciphertext gives the ciphertext to the servers, requesting a decryption share. The label is embedded in the ciphertext, and so each server can make its own decision as to the appropriateness of generating a decryption share. The user can verify the validity of the shares using the given verification key. When the user collects valid shares from at least k servers, he can apply a combining algorithm to obtain the decryption.

More formally, a threshold cryptosystem consists of the following algorithms:

- A *probabilistic key generation algorithm* **G** that takes as input a security parameter Λ, the number $n \geq 1$ of decryption severs, and the threshold parameter k ($1 \leq k \leq n$); it outputs
$$(PK, VK, \vec{SK}) = \mathbf{G}(\Lambda, n, k),$$
where PK is the *public encryption key*, VK is the *public verification key*, and $\vec{SK} = (SK_1, \ldots, SK_n)$ is the list of *private keys*.
- A *probabilistic encryption algorithm* **E** that takes as input the public key PK and a cleartext m, and a label L, and outputs a ciphertext $\psi = \mathbf{E}(PK, m, L)$.
- A *label extraction algorithm* **L** which takes as input a ciphertext ψ, and outputs a label $L = \mathbf{L}(\psi)$.
- A *probabilistic decryption share generation algorithm* **D** that takes as input a private key SK_i and a ciphertext ψ, and outputs a *decryption share* $\sigma = \mathbf{D}(SK_i, \psi)$.
- A *share verification algorithm* **V** that takes as input the public verification key VK, a ciphertext ψ, and decryption share σ, and outputs $\mathbf{V}(VK, \psi, \sigma) \in \{0, 1\}$.
- A *combining algorithm* **C** that takes as input the public verification key VK, a ciphertext ψ, and a set of decryption shares, and outputs a cleartext $m = \mathbf{C}(VK, \psi, S)$. The combining algorithm is also allowed to output a special "?" symbol that is distinct from all possible cleartext messages.

All of these algorithms should run in time polynomial in the length of their inputs (with the convention that inputs to **G** are encoded in unary).

Before going further, we introduce some further conventions.

- We assume that the private key SK_i encodes the index i of server P_i in some canonical way, and we say that "SK_i belongs to server P_i."
- We assume that a decryption share σ encodes in some canonical way the index of the server that (supposedly) created it. If this index is i, then we say that "σ belongs to sever P_i."
- We call a decryption share σ a *genuine* decryption share of ψ if it is a possible output of $\mathbf{D}(SK_i, \psi)$ for some $1 \leq i \leq n$.
- We call a decryption share σ a *valid* decryption share of ψ if $\mathbf{V}(VK, \psi, \sigma) = 1$.
- We say a set S of decryption shares is *full* if it contains k shares, no two of which belong to the same server.

There are a number of basic consistency conditions that should hold. For any output $(PK, VK, \vec{SK})$ of $\mathbf{G}(\Lambda, n, k)$, the following conditions should hold:

- *Correctness of label extraction*. For any message m, any label L, and any output ψ of $\mathbf{E}(PK, m, L)$, we have $\mathbf{L}(\psi) = L$.

  This condition merely ensures that the encryption algorithm embeds the label in the ciphertext in a canonical way.

- *Completeness of share verification*. Any genuine decryption share of a ciphertext ψ is also a valid decryption share of ψ.
- *Correctness of decryption*. Given

— any plaintext m and label L,
— any output ψ of $\mathbf{E}(PK, m, L)$,
— any full set S of genuine decryption shares of ψ,
we have $\mathbf{C}(VK, \psi, S) = m$.

The two basic properties that we want a threshold cryptosystem to have are *security against chosen ciphertext attack* and *consistency of decryptions*.

Security against chosen ciphertext attack means that any polynomial time adversary has a negligible advantage in the following game.

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 and verification key as well.
A3 The adversary interacts with the uncorrupted decryption servers in an arbitrary fashion, feeding them ciphertexts ψ, and obtaining decryption shares.
A4 The adversary chooses two cleartexts m_0 and m_1 (of the same length) and a label L. These are given to an "encryption oracle" that chooses $b \in \{0, 1\}$ at random, and gives the "target" ciphertext $\psi' = \mathbf{E}(PK, m_b, L)$ to the adversary.
A5 The adversary continues to interact with the uncorrupted servers, feeding them ciphertexts $\psi \neq \psi'$.
A6 At the end of the game, the adversary outputs $b' \in \{0, 1\}$.

The adversary's advantage is defined to be the absolute difference between $1/2$ and the probability that $b' = b$.

We assume the adversary runs in time polynomial in a given security parameter Λ. For technical reasons, it is convenient to assume that the adversary's running time is strictly polynomial bounded; i.e., it always halts after a polynomially bounded number of steps, regardless of its coin tosses, and regardless of the outputs of the key generation algorithm, the uncorrupted decryption servers, and the encryption oracle. The adversary is allowed to choose n and k as he likes, but their values must be bounded by a fixed polynomial in Λ, and may also have to satisfy further constraints imposed by a particular cryptosystem. When we say that a quantity, such as the adversary's advantage, is negligible, this means that as a function of Λ, it is less than than $1/Q(\Lambda)$, for any fixed polynomial in Q, and for sufficiently large Λ. To say that a quantity is non-negligible means that there is a polynomial $Q(\Lambda)$ such that for infinitely many Λ, the quantity is at least $1/Q(\Lambda)$.

Consistency of decryptions means that any polynomial time adversary has a negligible chance of winning the following game. The adversary interacts with the system exactly as in steps A1–A3 above. The adversary wins this game if he can produce a ciphertext ψ and two full sets S, S' of valid decryption shares such that $\mathbf{C}(VK, \psi, S) \neq \mathbf{C}(VK, \psi, S')$.

4. Basic Tools

4.1. *Threshold Secret Sharing*

Let q be a prime, and $1 \leq k \leq n < q$. Shamir's [Sh1] k out of n secret sharing scheme over $\mathbf{Z}_q$ works as follows. We have a 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 $1 \leq i \leq n$, let $x_i = F(i) \in \mathbf{Z}_q$ be the ith share of x. Just for notational purposes, we denote x as its own 0th share, so we have $x = x_0 = f_0$.

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 $S \subset \mathbf{Z}_q$ of cardinality k, any $i \in \mathbf{Z}_q$, and any $j \in S$, there exists an easy-to-compute element $\lambda_{ij}^S \in \mathbf{Z}_q$, such that

$$F(i) = \sum_{j \in S} \lambda_{ij}^S x_{i_j}.$$

4.2. *Intractability Assumptions*

Let G be a group of prime order q, generated by an element $g \in G$.

The *computational Diffie–Hellman problem* is this: given g^x and g^y for random $x, y \in \mathbf{Z}_q$, compute g^{xy}.

The *decisional Diffie–Hellman problem* is this: given a tuple that is either of the form (g^x, g^y, g^{xy}) or (g^x, g^y, g^z), where $x, y, z \in \mathbf{Z}_q$ are random, 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.

Triples of the form (g^x, g^y, g^{xy}) are called *Diffie–Hellman triples* (with respect to the base g).

4.3. *Zero-Knowledge Proof of Discrete Logarithm Identities*

Let G be a group of prime order q with generators $g, \bar{g}$. Let $\mathtt{EDLog}_{g,\bar{g}}$ 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 $\mathtt{EDLog}_{g,\bar{g}}$. 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 $\mathtt{EDLog}_{g,\bar{g}}$, due to Chaum and Pedersen [CP]. Although it also happens to be a proof of knowledge we do not use that property in our schemes.

Let $(u, \bar{u}) \in \mathtt{EDLog}_{g,\bar{g}}$ 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 $1/q$. For completeness, we recall the argument. Suppose $(u, \bar{u}) \notin \mathtt{EDLog}_{g,\bar{g}}$. That is, $u = g^r$ and $\bar{u} = \bar{g}^{r'}$, with $r \neq r'$. Suppose a cheating prover presents $(u, \bar{u})$ to a verifier, along with a pair $(w, \bar{w})$, where $w = g^s$ and $\bar{w} = \bar{g}^{s'}$. Now, if the verifier is to accept, then we must have that $g^f = wu^e$ and $\bar{g}^f = \bar{w}\bar{u}^e$. This implies that $(s - s') + e(r - r') = 0$. So, since $r - r' \neq 0$, there is at most one challenge to which the cheating prover can hope to respond, and the verifier generates this challenge with probability $1/q$. Actually, it is evident from this argument that a stronger soundness condition holds: the verifier will accept with at most probability $1/q$ if either $(u, \bar{u}) \notin \mathtt{EDLog}_{g,\bar{g}}$ or $(w, \bar{w}) \notin \mathtt{EDLog}_{g,\bar{g}}$.

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 oracle model, this becomes a non-interactive zero-knowledge proof of language membership.

Note that if $\bar{g}$ is not a generator for G, i.e., $\bar{g} = 1$, the above proof system can still be used to ensure that $\bar{u} = 1$. Thus, we can view the above proof system more generally as a proof that $(\bar{g}, u, \bar{u})$ is a Diffie–Hellman triple, where $\bar{g}$ is an arbitrary element of G. It is also easily seen that the above proof system ensures that $(\bar{g}, w, \bar{w})$ is also a Diffie–Hellman triple.

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:

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

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 *PK* consists of a description of the group G, along with the group elements g and h. The public verification key *VK* consists of the public key *PK*, along with the tuple $(h_1, \ldots, h_n)$ of group elements. For $1 \leq i \leq n$, the secret key SK_i consists of the public key *PK* along with the index i and the value $x_i \in \mathbf{Z}_q$.

To be technically complete, the key generation algorithm takes a security parameter Λ as input. The security parameter is used in selecting an appropriate group.

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

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

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

Note that with overwhelming probability, $\bar{g} \neq 1$, i.e., $\bar{g}$ generates G. In addition to an ordinary ElGamal encryption, consisting of the c and u, the above ciphertext also includes the group element $\bar{u}$, along with a non-interactive proof, consisting of e and f, that $\log_g u = \log_{\bar{g}} \bar{u}$.

Label extraction. Given an appropriately encoded ciphertext $\psi = (c, L, u, \bar{u}, e, f)$, the label extraction algorithm simply outputs L.

Decryption. Decryption server i does the following, given ciphertext $\psi = (c, L, u, \bar{u}, e, f)$. It checks if

$$e = H_3(\bar{g}, \bar{u}, \bar{w}), \qquad \text{where} \quad w = g^f/u^e, \quad \bar{g} = H_2(c, L, u, w), \quad \bar{w} = \bar{g}^f/\bar{u}^e. \tag{1}$$

If this condition does not hold, it outputs $(i, \text{“?”})$. Otherwise, it proceeds as follows. It chooses $s_i \in \mathbf{Z}_q$ at random, and computes

$$u_i = u^{x_i}, \quad \hat{u}_i = u^{s_i}, \quad \hat{h}_i = g^{s_i}, \quad e_i = H_4(u_i, \hat{u}_i, \hat{h}_i), \quad f_i = s_i + x_i e_i. \tag{2}$$

Its output is (i, u_i, e_i, f_i).

Note that the check (1) verifies the non-interactive proof that $\log_g u = \log_{\bar{g}} \bar{u}$. Also, the decryption share includes a non-interactive proof that (u, h_i, u_i) is a Diffie–Hellman triple, i.e., that $u_i = u^{x_i}$. This is needed to ensure consistency of decryption.

Share verification. The share verification algorithm is given the verification key *VK*, a ciphertext ψ, and a decryption share belonging to some server i. The verification algorithm first tests if (1) holds. If this does not hold, then a decryption share is valid if and only if it is of the form $(i, \text{“?”})$. Otherwise, the share is considered valid if and only if it is of the form (i, u_i, e_i, f_i), and

$$e_i = H_4(u_i, \hat{u}_i, \hat{h}_i), \qquad \text{where} \quad \hat{u}_i = u^{f_i}/u_i^{e_i}, \quad \hat{h}_i = g^{f_i}/h_i^{e_i}. \tag{3}$$

Note that the check (3) ensures that (u, h_i, u_i) is a Diffie–Hellman triple.

Combining shares. The share combining algorithm takes as input the verification key *VK*, a ciphertext ψ, and a full set of valid decryption shares of ψ. If the test (1) does not hold, then we output “?” (all the decryption shares are of the form $(i, \text{“?”})$ in this case). So we can assume that the set of decryption shares is of the form

$$\{(i, u_i, e_i, f_i) \colon i \in S\},$$

where $S \subset \{1, \ldots, n\}$ has cardinality k. Then, using the notation defined in Section 4.1, the recovery algorithm outputs

$$m = H_1\left(\prod_{i \in S} u_i^{\lambda_{0i}^S}\right) \oplus c.$$

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

Proof. We show how to use an adversary that can guess bit b in game A to solve the computational Diffie–Hellman problem. It is clear that if the adversary is able 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 accurate, but it does not matter: we already solved the computational Diffie–Hellman problem. Actually, the output of our algorithm is simply a list of all points at which H_1 was queried, which with non-negligible probability will contain the solution to the computational Diffie–Hellman problem. The Diffie–Hellman self-corrector in [Sh2] can be used to transform this into an algorithm that outputs a single, correct solution to the computational Diffie–Hellman problem.

We now give the details of the simulation. Let $\alpha, \beta \in G$ be random elements in G for which we want to solve the computational 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 oracles. 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 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 $h = \alpha$. Note that with overwhelming probability, $h \neq 1$, and we assume this in what follows. 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 non-corrupt decryption servers.

We deal first with the encryption oracle. The adversary gives a label L' and two messages, m_0 and m_1, to the encryption oracle. We ignore the messages completely. Instead, we simply choose $c' \in \{0, 1\}^l$ and $t', e', f' \in \mathbf{Z}_q$ at random. We then set

$$u' = \beta, \qquad \bar{g}' = g^{t'}, \qquad \bar{u}' = (u')^{t'}, \qquad w' = g^{f'}/(u')^{e'}, \qquad \bar{w}' = (\bar{g}')^{f'}/(\bar{u}')^{e'}.$$

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

It is easily verified that this backpatching is allowable, at least with overwhelming probability. Also, one sees that u', $\bar{g}'$, and $\bar{u}'$ have the right distribution; namely, $(u', \bar{g}', \bar{u}')$ is a random Diffie–Hellman triple. The rest is just a standard zero-knowledge simulation.

Thus, the simulation is statistically close to perfect, as long as the adversary does not query H_1 at the point $(u')^{\log_g h} = \gamma$. Note that the simulator cannot detect when and if this event occurs—that would be tantamount to solving the decisional Diffie–Hellman problem. Nevertheless, if this does occur, we will already have a solution to the given computational Diffie–Hellman problem in our list of inputs to the H_1 oracle, so we do not care if the view presented to the adversary by the simulator is inaccurate after this occurs.

We next deal with the simulation of the uncorrupted decryption servers. First, whenever the adversary queries H_2 at a point other than (c', L', u', 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). Note that $t \neq 0$ with overwhelming probability, and we assume this in what follows.

Now suppose P_i is given a valid ciphertext $\psi \neq \psi'$, where $\psi = (c, L, u, \bar{u}, e, f)$. Now, $(c, L, u, \bar{u}, e, f)$ determines via the validity condition (1) corresponding variables $\bar{g}, w, \bar{w}$.

We first argue that we can assume that $(c, L, u, w) \neq (c', L', u', w')$. On the contrary, suppose that $(c, L, u, w) = (c', L', u', w')$. Then of course $\bar{g} = \bar{g}'$. However, with overwhelming probability, we must also have $(\bar{u}, \bar{w}) = (\bar{u}', \bar{w}')$; this follows immediately from the strong soundness condition discussed in Section 4.3. It then follows that $e = e'$, since $e = H(\bar{g}, \bar{u}, \bar{w}) = e'$. From this, it follows that $f = f'$, since the equation $w = g^f/u^e$ uniquely determines f, once w, u, and e are determined. This then contradicts our assumption that $\psi \neq \psi'$.

So assume $(c, L, u, w) \neq (c', L', u', w')$. 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 $t \neq 0$ 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. However, by the soundness of the proof that $\log_g u = \log_{\bar{g}} \bar{u}$, we can assume that $\bar{u} = \bar{g}^r$, and hence $(\bar{u})^{1/t} = (\bar{g})^{r/t} = h^r$.

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 (u, h_i, u_i) is a Diffie–Hellman triple. However, u_i can be computed by the simulator as $u_i = (\bar{u})^{\lambda_{i0}/t} \prod_{j=1}^{k-1} u^{x_j \lambda_{ij}}$. Once we have u_i, we can readily produce a zero-knowledge simulation of the proof that (u, h_i, u_i) is a Diffie–Hellman triple, backpatching H_4 as necessary.

That completes the proof of Theorem 1. □

Theorem 2. *In the random oracle model, the* TDH1 *cryptosystem satisfies the consistent decryption property.*

This theorem follows immediately from the soundness of the equality of discrete logarithm protocol in Section 4.3.

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:

$$H_1\colon G \to \{0,1\}^l, \qquad H_2\colon \{0,1\}^l \times \{0,1\}^l \times G^4 \to \mathbf{Z}_q, \qquad H_4\colon G^3 \to \mathbf{Z}_q.$$

Key generation. Same as for TDH1, except that a random generator $\bar{g} \in G$ 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

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

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

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

Label extraction. Same as in TDH1.

Decryption. Decryption server i does the following, given ciphertext $(c, L, u, \bar{u}, e, f)$. It checks if

$$e = H_2(c, L, u, w, \bar{u}, \bar{w}), \qquad \text{where} \quad w = g^f/u^e, \quad \bar{w} = \bar{g}^f/\bar{u}^e. \tag{4}$$

If this condition does not hold, it outputs $(i, \text{“?”})$. Otherwise, it computes u_i, e_i, f_i as in (2), creating an output (i, u_i, e_i, f_i).

Share Verification. Same as for TDH1, except that we use the test (4), instead of the test (1).

Combining shares. Same as for TDH1, except that we use the test (4), instead of the test (1).

Theorem 3. *In the random oracle model, the* TDH2 *cryptosystem is secure against chosen ciphertext attack, assuming the decisional Diffie–Hellman problem in G is hard.*

Proof. Again, the proof is by reduction, and we assume the adversary queries, with non-negligible probability, H_1 at 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 decisional Diffie–Hellman 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 non-negligible 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 $\bar{g} = h^t\ (= g^{xt})$. Note that with overwhelming probability, we have $h \neq 1$ and $t \neq 0$, and we assume this in what follows.

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

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 $t \neq 0$ with $\bar{g} = h^t$, and so given a valid ciphertext $(c, L, u, \bar{u}, e, f) \neq (c', L', u', \bar{u}', e', f')$, 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 $(h, \bar{g}, u', \bar{u}')$. In the case where (α, β, γ) is drawn from the Diffie–Hellman triple distribution, $(h, \bar{g}, u', \bar{u}')$ is (statistically indistinguishable from) a random element of G^4, subject to the condition $\bar{g} \neq 1$ and $\log_g u' = \log_{\bar{g}} \bar{u}'$. In the case where (α, β, γ) is a random triple, $(h, \bar{g}, u', \bar{u}')$ is (statistically indistinguishable from) a random element of G^4, subject to the condition that $\bar{g} \neq 1$. In either case, γ is determined by $\gamma = (\bar{u}')^{\log_{\bar{g}} h}$; moreover, if (α, β, γ) is a Diffie–Hellman triple, then the relation $\gamma = (u')^{\log_g h}$ 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 statistically close to perfect until the adversary queries H_1 at $\gamma = (u')^{\log_g h}$, 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 accurate up to this point, this happens with non-negligible 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 non-negligible probability. As mentioned above, $(h, \bar{g}, u', \bar{u}')$ is essentially just a random element in G^4 (with $\bar{g} \neq 1$). The other random variables e', f', w', and

$\bar{w}'$ are also just random, subject to relations that make the "proof" that $\log_g u' = \log_{\bar{g}} \bar{u}'$ look legitimate; in fact, the relation $\log_g u' = \log_{\bar{g}} \bar{u}'$ 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 $(\bar{u}')^{\log_{\bar{g}} h}$ given random $(h, \bar{g}, u', \bar{u}') \in G^4$.

As we show below, we can use this adversary to solve the following "inverse" Diffie–Hellman problem with the non-negligible probability: given random $\alpha' = g^t$ and $\beta' = g^v$, compute $\gamma' = g^{v/t}$. It is easy to see that the "inverse" Diffie–Hellman problem is random self-reducible, just like the computational Diffie–Hellman problem. Also, the self-corrector for the computational Diffie–Hellman problem in [Sh2] can be easily modified to yield a self-corrector for the "inverse" Diffie–Hellman problem. It is easy to see that this "inverse" Diffie–Hellman problem is equivalent (under polynomial-time reduction) to the computational Diffie–Hellman problem. In particular, we can solve an instance of the computational Diffie–Hellman by making two queries to an "inverse" Diffie–Hellman oracle.

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

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

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

So, if the adversary causes the first simulator on a random triple to output 1 with non-negligible probability, then this same adversary causes this new simulator to output a list containing the desired solution γ' to the given instance of the "inverse" Diffie–Hellman problem.

That completes the proof of Theorem 3. □

Theorem 4. *In the random oracle model, the* TDH2 *cryptosystem satisfies the consistent decryption property.*

Just as for THD1, this theorem follows immediately from the soundness of the equality of discrete logarithm protocol in Section 4.3.

7. Implementation Issues

To implement these schemes, one has to choose concrete hash functions. This is relatively straightforward, but see [BR1] 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 [BGMW], [LL2]. 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 oracle model assuming, respectively, that the computational and decisional Diffie–Hellman 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 Section 2.7, which is not known to be secure in the random oracle 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 against chosen ciphertext attack (even using the random oracle model); (3) find a practical threshold cryptosystem that is provably secure against chosen ciphertext attack, without using the random oracle model; to date, the most practical such schemes known [CG1], [Ab], [JL] require either synchronized interaction or a large number of pre-shared secrets, which makes them much less practical than the schemes presented here.

References

[Ab] M. Abe. Robust distributed multiplication without interaction. In *Advances in Cryptology – Crypto* '99, pages 130–147, 1999.

[ABR1] M. Abdalla, M. Bellare, and P. Rogaway. DHAES: an encryption scheme based on the Diffie–Hellma problem. Cryptology ePrint Archive, Report 1999/007, `http://eprint.iacr.org`.

[ABR2] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie–Hellman assumptions and an analysis of DHIES. In *Topics in Cryptology – CT-RSA* 2001, pages 143–158. LNCS 2045. Springer-Verlag, Berlin, 2001.

[ASW] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000. Extended abstract in *Advances in Cryptology – Eurocrypt* '98.

[BDMP] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive zero knowledge. *SIAM Journal on Computing*, 6(4):1084–1118, 1991.

[BDPR] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology – Crypto* '98, pages 26–45, 1998.

[BG] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *Advances in Cryptology – Crypto* '92, pages 390–420. LNCS 740. Springer-Verlag, Berlin, 1993.

[BGMW] 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.

[BLK] J. Baek, B. Lee, and K. Kim. Secure length-saving ElGamal encryption under the computational Diffie–Hellman assumption. In *Proc*. 5*th Australian Conference on Information, Security, and Privacy*, pages 49–58, 2000.

[BMP] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie–Hellman. In *Advances in Cryptology – Eurocrypt* 2000, pages 156–171, 2000.

[Bo] C. Boyd. Digital multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Clarendon Press, Oxford, 1986.

[BR1] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc*. *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR2] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology – Eurocrypt* '94, pages 92–111, 1994.

[BR3] M. Bellare and P. Rogaway. The exact security of digital signatures: how to sign with RSA and Rabin. In *Advances in Cryptology – Eurocrypt* '96, pages 399–416, 1996.

[BS] M. Bellare and A. Sahai. Non-malleable encryption: equivalence between two notions, and an indistinguishability-based characterization. In *Advances in Cryptology – Crypto* '99, pages 519–536, 1999.

[CG1] R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – Eurocrypt* '99, pages 90–106, 1999.

[CG2] D. Catalano and R. Gennaro. New efficient and secure protocols for verifiable signature sharing and other applications. In *Advances in Cryptology – Crypto* '99, pages 105–120, 1999.

[CGH] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisted. In *Proc*. 30*th Annual ACM Symposium on Theory of Computing*, pages 209–218, 1998.

[CKPS] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology – Crypto* 2001, pages 524–541, 2001. Full length version available as Cryptology ePrint Archive, Report 2001/006, `http://eprint.iacr.org`.

[CKS] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. In *Proc*. 19*th ACM Symposium on Principles of Distributed Computing*, pages 123–132, 2000.

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

[CS] 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.

[Da] I. Damgård. Towards practical public key cryptosystems secure against chosen ciphertext attacks. In *Advances in Cryptology – Crypto* '91, pages 445–456, 1991.

[DDFY] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proc*. 26*th Annual ACM Symposium on Theory of Computing*, pages 522–533, 1994.

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

[De] Y. Desmedt. Society and group oriented cryptography: a new concept. In *Advances in Cryptology – Crypto* '87, pages 120–127, 1987.

[DF] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology – Crypto* '89, pages 307–315, 1989.

[DH] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[DK] Y. Desmedt and K. Kurosawa. How to break a practical MIX and design a new one. In *Advances in Cryptology – Crypto* 2000, pages 557–572, 2000.

[FFS] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1:77–94, 1988.

[FO] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology – Crypto '*99, pages 537–554, 1999.

[FOPS] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *Advances in Cryptology – Crypto* 2001, pages 260–274, 2001.

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

[FS] 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. LNCS 263. Springer-Verlag, Berlin, 1987.

[FY] Y. Frankel and M. Yung. Cryptanalysis of immunized LL public key systems. In *Advances in Cryptology – Crypto '*95, pages 287–296, 1995.

[GJKR] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology – Eurocrypt '*99, pages 295–301, 1999.

[Ja] M. Jakobsson. A practical MIX. In *Advances in Cryptology – Eurocrypt '*98, pages 448–461, 1998.

[JL] S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: introducing concurrency, removing erasures. In *Advances in Cryptology – Eurocrypt* 2000, pages 221–242, 2000.

[LL1] 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.

[LL2] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology – Crypto '*94, pages 95–107, 1994.

[NY] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc*. 22*nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.

[Po] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In *Proc*. 2000 *International Workshop on Practice and Theory in Public Key Cryptography* (*PKC* 2000), pages 129–146, 2000.

[PS1] D. Pointcheval and J. Stern. Provably secure blind signature schemes. In *Advances in Cryptology – Asiacrypt '*96, pages 252–265, 1996.

[PS2] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – Eurocrypt '*96, pages 387–398, 1996.

[RB] M. Reiter and K. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems*, 16:986–1009, 1994.

[RS] 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.

[RSA] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

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

[SG] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology – Eurocrypt '*98, pages 1–16, 1998.

[Sh1] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[Sh2] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – Eurocrypt '*97, pages 256–266, 1997.

[Sh3] V. Shoup. Practical threshold signatures. In *Advances in Cryptology – Eurocrypt* 2000, pages 207–220, 2000.

[Sh4] V. Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *Advances in Cryptology – Eurocrypt* 2000, pages 275–288, 2000.

[Sh5] V. Shoup. OAEP reconsidered. In *Advances in Cryptology – Crypto* 2001, pages 239–259, 2001. Full length version available as Cryptology ePrint Archive, Report 2000/060, `http://eprint.iacr.org`.

[TY] Y. Tsiounis and M. Yung. On the security of ElGamal based encryption. In *Proc*. *PKC '*98, pages 117–134, 1998.

[ZS] 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.