Certificate-Based Encryption and the Certificate Revocation Problem

Craig Gentry

DoCoMo USA Labs
cgentry@docomolabs-usa.com

Abstract. We introduce the notion of certificate-based encryption. In this model, a certificate – or, more generally, a signature – acts not only as a certificate but also as a decryption key. To decrypt a message, a keyholder needs both its secret key and an up-to-date certificate from its CA (or a signature from an authorizer). Certificate-based encryption combines the best aspects of identity-based encryption (implicit certification) and public key encryption (no escrow). We demonstrate how certificate-based encryption can be used to construct an efficient PKI requiring less infrastructure than previous proposals, including Micali's Novomodo, Naor-Nissim and Aiello-Lodha-Ostrovsky.

1 Introduction

A (digital) certificate is a signature by a trusted certificate authority (CA) that securely binds together several quantities. Typically, these quantities include at least the name of a user U and its public key PK. Often, the CA includes a serial number SN (to simplify its management of the certificates), as well as the certificate's issue date D_1 and expiration date D_2. By issuing $Sig_{CA}(U, PK, SN, D_1, D_2)$, the CA basically attests to its belief that PK is (and will be) user U's authentic public key from the current date D_1 to the future date D_2.

Since CAs cannot tell the future, circumstances may require a certificate to be revoked before its intended expiration date. For example, if a user accidentally reveals its secret key or an attacker actively compromises it, the user itself may request revocation of its certificate. Alternatively, the user's company may request revocation if the user leaves the company or changes position and is no longer entitled to use the key.

If a certificate is revocable, then third parties cannot rely on that certificate unless the CA distributes *certificate status* information indicating whether the certificate is currently valid. This certificate status information must be fresh – e.g., to within a day. Moreover, it must be widely distributed (to all relying parties). Distributing large amounts of fresh certification information is the "certificate revocation problem." Solving this problem seems to require a lot of *infrastructure*, and the apparent need for this infrastructure is often cited as a reason against widespread implementation of public-key cryptography.

1.1 Some Previous Solutions to the Certificate Revocation Problem

The most well-known – and a very inefficient – public-key infrastructure (PKI) proposal is the certificate revocation list (CRL). A CRL is simply a list of certificates that have been revoked before their intended expiration date. The CA issues this list periodically, together with its signature. Since the CA will likely revoke many of its certificates – say, 10% if they are issued with an intended validity period of one year [15] – the CRL will be quite long if the CA has many clients. Nonetheless, the complete list must be transmitted to any party that wants to perform a certificate status check. There are refinements to this approach, such as delta CRLs that list only those certificates that have been revoked since the CA's last update, but the transmission costs, and the infrastructural costs necessary to enable the transmission, are still quite high.

Another proposal is called the Online Certificate Status Protocol (OCSP). The CA responds to a certificate status query by generating (online) a fresh signature on the certificate's current status. This reduces transmission costs to a single signature per query, but it substantially increases computation costs. It also decreases security: if the CA is centralized, it becomes highly vulnerable to denial-of-service (DoS) attacks; if it is distributed and each server has its own secret key, then compromising any server compromises the entire system [15].

A much more promising line of research, which has not received enough attention from industry, was initiated by Micali [14], [15]. (See also [16], [1], [12].) Similar to previous PKI proposals, Micali's "Novomodo" system involves a CA, one or more directories (to distribute the certification information), and the users. However, it achieves better efficiency than CRLs and OCSP, without sacrifices in security.

Micali's basic scheme (slightly oversimplified) is as follows. For each client, the CA chooses a random 160-bit value X_0 and repeatedly applies a public one-way hash function to it to obtain the 160-bit value X_n, where $X_i = H(X_{i-1})$.[1] The CA includes X_n in the client's certificate: $Sig_{CA}(U, PK, SN, D_1, D_2, X_n)$. If U's certificate is still valid on the ith day after issuance, the CA sends the value of X_{n-i} to the directories; otherwise, it does not. In the former case, third party T can verify that U's certificate is still valid by querying a directory, and then checking that $H^i(X_{n-i})$ (i times) equals X_n, the value embedded in U's certificate.

The advantage of Novomodo over a CRL-based system is that a directory's response to a certificate status query is concise – just 160 bits (if T has cached $Sig_{CA}(U, \ldots, X_n)$). The length of a CRL, on the other hand, grows with the number of certificates that have been revoked. Novomodo has several advantages over OCSP. First, since hashing is computationally cheaper than signing, the CA's computational load in Novomodo is typically much lower. Second, unlike the distributed components of an OCSP CA, the directories in Novomodo need not be trusted. Instead of producing signatures that are relied on by third parties,

---

1 For example, the one-way hash may be SHA1, and n may equal 365 if the intended validity period of a certificate is one year and we want a freshness of one day.

the directories merely distribute hash preimages sent by the CA (which they cannot produce on their own). Third, the lack of online computation by the directories makes Novomodo less susceptible to DoS attacks. Finally, although OCSP already has fairly low directory-to-user communication (one constant-length signature per query), Novomodo's is typically even lower, since public-key signatures are typically longer than 160 bits.

Following Micali's proposal, Naor-Nissim [16] and Aiello-Lodha-Ostrovsky [1] proposed different hash-based systems with different computation and communication tradeoffs. Essentially, both of these proposals use binary trees to reduce the CA's computation even further, as well as to reduce CA-directory communication.

1.2 The Problem with Third-Party Queries

Novomodo and related proposals are improvements over CRLs and OCSP. However, the infrastructural requirements of these approaches can vary dramatically, depending on how users use their keys. Suppose that users' keys are used only to generate and verify signatures (never to encrypt and decrypt). In this case, we do not need any infrastructure to deal with *third-party queries* – i.e., queries by one party on a different party's certificate status. Why not? – because a signer can simply send its proof of certificate status to the verifier simultaneously with its signature. In other words, when a protocol allows a client to furnish its certificate status, no third-party queries are necessary. We only need infrastructure that allows each client to obtain its *own* proof of certification.

However, the situation may be different if users use their keys for encryption and decryption. In this case, third party T must obtain U's certification status *before* sending an encrypted message to U. Getting this information directly from U may not be an option in high-latency applications like email (where U might not respond promptly), or in applications where it is preferable to avoid the extra round trip. Conceivably, T could obtain U's certificate status from some other source – perhaps a server affiliated with U but not with the CA – but this seems *ad hoc*, and it still requires an extra round trip. Novomodo and related proposals would address this situation by allowing T to make a third-party query.

We would like to eliminate, or at least strongly disincentivize, third-party queries for several reasons. First, since third party queries can come from anywhere and concern any client, every certificate server in the system must be able to ascertain the certificate status of every client in the system. The situation is much cleaner if third-party queries are eliminated. Each server only needs to have certification proofs for the clients that it serves. Moreover, these proofs could be "pushed" to clients, and multicast might be used to dramatically reduce the CA's transmission costs. Second, third party queries multiply the query processing costs of the CA and/or its servers. For example, suppose each client queries the certification status of 10 other clients per day. Then, the system must process $10N$ queries (where N is the number of clients), rather than just N (if each client only retrieves its own daily proof of certification). Third, nonclient queries are undesirable from a business model perspective: if T is not a client of U's CA, what incentive does the CA have to give T fresh certificate status information? Finally, there is a security consideration: if the CA must respond to queries from nonclients, it becomes more susceptible to DoS attacks. In summary, eliminating third-party queries allows a CA to reduce its infrastructural costs, simplify its business model and enhance security.

So, how can we eliminate third-party queries without constraining how users use their keys? We will use an approach based on *implicit certification* [2] – i.e., where T, without obtaining explicit information other than U's public key and the parameters of U's CA, can encrypt its message to U so that U can decrypt only if it is currently certified. [3] This will allow us to enjoy the infrastructural benefits of eliminating third-party queries, regardless of how users use their keys.

1.3 Identity-Based Encryption

One way to achieve implicit certification in the encryption context is identity-based encryption (IBE). Shamir [18] originated the concept of identity-based cryptography in 1984, describing an identity-based signature scheme in the same article. However, fully practical IBE schemes have been discovered only recently. We briefly review IBE below.

An IBE scheme uses a trusted third party called a Private Key Generator (PKG). To set up, the PKG generates a master secret s, and it publishes certain system parameters *params* that include a public key that masks s. The PKG may have many clients. Each client has some ID, which is simply a string that identifies it – e.g., the client's email address. The main algorithms in IBE are as follows.

1. Private key generation: For a given string ID, PKG uses (s, *params*, ID) to compute the corresponding private key d_{ID}.
2. Encryption: Sender uses (*params*, ID, M) to compute C, ciphertext for M.
3. Decryption: Client uses (*params*, d_{ID}, C) to recover M.

Notice that a client's public key, ID, can be arbitrary, but standardizing its format allows senders to "guess" the client's public key rather than obtain it from the client or a directory. Revocation is handled by including the $DATE$ (for example) as part of ID, so that keys expire after one day. Notice also that the PKG generates the private keys of all of its clients. This has several consequences. The most important, for our purposes, is that *certification is implicit*: a client can decrypt only if the PKG has given a private key to it (in effect, certifying the client). However, there are two negative consequences: 1) private key *escrow* is inherent in this system – i.e., a PKG can easily decrypt its clients' messages; and

[2] This should not be confused with "implicit certification" in the context of self-certified keys. In that context, T still must obtain explicit U-specific information following each certification update.

[3] In some of our schemes, T must also obtain the *long-lived* certificate for U's public key. (But T never needs to obtain U's *fresh* certificate status information.)

2) the PKG must send client private keys over *secure channels*, making private key distribution difficult.

These two disadvantages of IBE, particularly escrow, may be unacceptable for some applications, like email. Fortunately, we can get rid of both of them without sacrificing implicit certification. The basic idea is simple, and is the usual way of circumventing escrow: double encryption.

1.4 Our Results

We introduce the notion of *certificate-based encryption* (CBE), which combines public-key encryption (PKE) and IBE while preserving most of the advantages of each. As with PKE, each client generates its own public-key / secret-key pair and requests a certificate from the CA. The main difference is that the CA uses an IBE scheme to generate the certificate. This certificate has all of the functionality of a conventional PKI certificate – e.g., it can be used *explicitly* as proof of current certification (even of a signature key) – but it can also be used as a decryption key. This added functionality gives us implicit certification – Alice can doubly encrypt her message to Bob so that Bob needs both his personal secret key and an up-to-date certificate from his CA to decrypt. Implicit certification, in turn, allows us to eliminate third-party queries on certificate status. There is no escrow in CBE (since the CA does not know Bob's personal secret key), and there is no secret key distribution problem (since the CA's certificate need not be kept secret). We will introduce CBE more fully in Section 2, and describe a pairing-based CBE scheme secure against chosen-ciphertext attack in Section 3.

By itself, ordinary CBE becomes inefficient when the CA has a large number of clients (say, 250 million) and performs frequent certificate updates (say, hourly). Such a CA must issue about 225 million certificates per hour (62500 per second), assuming, as in [15], that about 10% of the CA's clients have been revoked. By current standards, such a CA would need considerable computational power.

In Sections 4 and 5, we describe how to refine basic CBE to construct an exceptionally efficient PKI. The scheme of Section 4 reduces the CA's computation through the use of *subset covers*. The CA embeds a serial number in each client's long-lived certificate that represents the position of the client's leaf in a binary tree. To reconfirm the validity of a client's certificate, the CA publishes the certificate / decryption key of an ancestor of the client leaf. Using this scheme, the CA only needs to compute an average of $R_{total} \log(N/R_{total})$ certificates, where N is the number of clients and R_{total} is the total number of clients whose certificates have been revoked but have not yet expired. Each certificate is constant length – as little as 160 bits – and can, if desired, also be used as explicit proof of certification. This scheme manages to achieve bandwidth-efficiency basically identical to a scheme described by Aiello, Lodha and Ostrovsky [1], but with CBE functionality. [4]

[4] Other tradeoffs are worth mentioning. Unlike ALO, the time need to verify an explicit certification proof in our scheme does not grow linearly with the number of time periods, but ALO might still be faster since it uses hash chains.

The scheme of Section 5 combines the use of subset covers with an incremental approach. Using incremental CBE, the CA can reduce its computation dramatically; the CA needs to compute only $R_{period} \log(N/R_{period})$ certificates per period, where R_{period} is the number of clients whose certificates have been revoked *in the period*.[5] Since $R_{hour} \approx .1N/(365 * 24) \approx 2850$, the CA only needs to compute an average of about 13 certificates per second even if updates are hourly. Using supersingular elliptic curves, this computation is quite reasonable – indeed, a single 1 GHz Pentium III PC can handle it easily – since each certificate generation essentially amounts to one elliptic curve point multiplication. Each client consolidates its periodic certificates simply by adding them together, and the consolidated certificate acts as the second decryption key. Of course, like the other CBE schemes, this scheme also eliminates third-party queries.

In Section 6, we describe various extensions of the CBE schemes, such as hierarchical CBE. Finally, we present a summary.

2 Certificate-Based Encryption

2.1 The Model

We now provide a formal security model for certificate-based encryption (CBE). The two main entities involved in CBE are a certifier and a client. Our definition of CBE is somewhat similar to that of strongly key-insulated encryption [10], but, among other differences, our model does not require a secure channel between the two entities. (See Appendix D of [4] for a discussion of this secure channel requirement.)

Definition 1. A *certificate-updating certificate-based encryption scheme* is a 6-tuple of algorithms $(\mathsf{Gen}_{\mathsf{IBE}}, \mathsf{Gen}_{\mathsf{PKE}}, \mathsf{Upd1}, \mathsf{Upd2}, \mathsf{Enc}, \mathsf{Dec})$ such that:

1. The probabilistic IBE key generation algorithm $\mathsf{Gen}_{\mathsf{IBE}}$ takes as input a security parameter 1^{k_1} and (optionally) the total number of time periods t. It returns SK_{IBE} (the certifier's master secret) and public parameters $params$ that include a public key PK_{IBE}, and the description of a string space $\mathcal{S}$.
2. The probabilistic PKE key generation algorithm $\mathsf{Gen}_{\mathsf{PKE}}$ takes as input a security parameter 1^{k_2} and (optionally) the total number of time periods t. It returns a secret key SK_{PKE} and public key PK_{PKE} (the client's secret and public keys).
3. At the start of time period i, the deterministic certifier update algorithm $\mathsf{Upd1}$ takes as input SK_{IBE}, $params$, i, string $s \in \mathcal{S}$ and PK_{PKE}. It returns $Cert'_i$, which is sent to the client.
4. At the start of time period i, the deterministic client update algorithm $\mathsf{Upd2}$ takes as input $params$, i, $Cert'_i$, and (optionally) $Cert_{i-1}$. It returns $Cert_i$.

[5] Aiello, Lodha and Ostrovsky also describe a scheme in which the CA issues $R_{period} \log(N/R_{period})$ certificates per period, but the size of a client's explicit proof of certification grows linearly with the number of time periods, whereas the length of the proof (a.k.a. the second decryption key) in our scheme tops out at $\log N$.

5. The probabilistic encryption algorithm Enc takes $(params, i, s, PK_{PKE}, M)$ as input, where M is a message. It returns a ciphertext C on message M intended for the client to decrypt using $Cert_i$ and SK_{PKE} (and possibly s).
6. The deterministic decryption algorithm Dec takes $(params, Cert_i, SK_{PKE}, C)$ as input in time period i. It returns either M or the special symbol $\perp$ indicating failure. We require $\mathsf{Dec}_{Cert_i,SK_{PKE},s}(\mathsf{Enc}_{i,s,PK_{IBE},PK_{PKE}}(M)) = M$ for the given $params$.

Remark 1. CBE does not necessarily have to be "certificate updating," and it can be useful for applications other than certificate management. In particular, it may be useful in other situations where authorization or access control is an issue. An encrypter can use CBE to encrypt its message so that the keyholder can decrypt only after it has obtained certain signatures from one or more authorizers on one or more messages.

Remark 2. It may seem strange that a certificate – or, more generally, a signature – can be used as a decryption key. However, as noted by Moni Naor [6], any IBE scheme immediately gives a public key signature scheme as follows: We set the signer's private key to be the master key in the IBE scheme. The signer signs M by computing the IBE decryption key d for $ID = M$. The verifier merely has to check that d correctly decrypts messages encrypted with $ID = M$; so, the verifier chooses a random message M', encrypts it with $ID = M$ using the IBE scheme, and then tries to to decrypt the resulting ciphertext with d. If the ciphertext decrypts correctly, the signature is considered valid. Thus, an IBE decryption key is also a signature (or a certificate). This certificate / decryption key can be verified like a signature as explicit proof of certification (even of signature keys), or it can be used as a means for enabling implicit certification in the encryption context, as described in the Introduction.

As should be clear from Definition 1, we model CBE essentially as a combination of PKE and IBE, where the client needs both its personal secret key and a certificate / decryption key from the CA to decrypt. The string s may include a message that the certifier "signs" – e.g., the certifier may sign $clientinfo = \langle clientname, PK_{PKE}\rangle$. (Notice that $clientinfo$ contains only long-lived information about the client; an encrypter need not know the client's current certificate status.) Depending on the scheme, s may include other information, such as the client's signature on its public key.

2.2 Security

Roughly speaking, we are primarily concerned about two different types of attacks: 1) by an uncertified client and 2) by the certifier. We want CBE ciphertexts to be secure against each of these entities, even though each basically has "half" of the secret information needed to decrypt. Accordingly, we define IND-CCA in terms of two different games. The adversary chooses one game to play. In Game 1, the adversary essentially assumes the role of an uncertified client. After proving knowledge of the secret key corresponding to its claimed public key, it can make Dec and $\mathsf{Upd1}$ queries. In Game 2, the adversary essentially assumes the role of the certifier. After proving knowledge of the master secret corresponding to its claimed *params*, it can make Dec queries. Roughly, we say that a CBE scheme is secure if no adversary can win either game.

Game 1: The challenger runs $\mathsf{Gen}_{\mathsf{IBE}}(1^{k_1}, t)$, and gives *params* to the adversary. The adversary then interleaves certification and decryption queries with a single challenge query. These queries are answered as follows:

– On certification query $(i, s, PK_{PKE}, SK_{PKE})$, the challenger checks that $s \in \mathcal{S}$ and that SK_{PKE} is the secret key corresponding to PK_{PKE}. If so, it runs $\mathsf{Upd1}$ and returns $Cert'_i$; else it returns $\perp$.
– On decryption query $(i, s, PK_{PKE}, SK_{PKE}, C)$, the challenger checks that $s \in \mathcal{S}$ and that SK_{PKE} is the secret key corresponding to PK_{PKE}. If so, it generates $Cert_i$ and outputs $\mathsf{Dec}_{Cert_i, SK_{PKE}, s}(C)$; else it returns $\perp$.
– On challenge query $(i', s', PK'_{PKE}, SK'_{PKE}, M_0, M_1)$, the challenger checks that $s' \in \mathcal{S}$ and that SK'_{PKE} is the secret key corresponding to PK'_{PKE}. If so, it chooses random bit b and returns $C' = \mathsf{Enc}_{i', s', PK_{IBE}, PK'_{PKE}}(M_b)$; else it returns $\perp$.

Eventually, the adversary outputs a guess $b' \in \{0, 1\}$. The adversary wins the game if $b' = b$ and (i', s', PK'_{PKE}, C') was not the subject of a valid decryption query after the challenge, and (i', s', PK'_{PKE}) was not the subject of any valid certification query. The adversary's advantage is defined to be the absolute value of the difference between 1/2 and its probability of winning.

Game 2: The challenger runs $\mathsf{Gen}_{\mathsf{PKE}}(1^{k_2}, t)$, and gives PK_{PKE} to the adversary. The adversary then interleaves decryption queries with a single challenge query. These queries are answered as follows:

– On decryption query $(i, s, params, SK_{IBE}, C)$, the challenger checks that $s \in \mathcal{S}$ and that SK_{IBE} is the secret key corresponding to *params*. If so, it generates $Cert_i$ and outputs $\mathsf{Dec}_{Cert_i, SK_{PKE}, s}(C)$; else it returns $\perp$.
– On challenge query $(i', s', params', SK'_{IBE}, M_0, M_1)$, the challenger checks that $s' \in \mathcal{S}$ and that SK'_{IBE} is the secret key corresponding to *params*′. If so, it chooses random bit b and returns $C' = \mathsf{Enc}_{i', s', PK'_{IBE}, PK_{PKE}}(M_b)$; else it returns $\perp$.

The adversary guesses $b' \in \{0, 1\}$ and wins if $b' = b$ and $(i', s', params', C')$ was not the subject of a valid decryption query after the challenge. The adversary's advantage is defined as above.

Definition 2. A certificate-updating certificate-based encryption scheme is *secure against adaptive chosen ciphertext attack* (IND-CBE-CCA) if no PPT adversary has non-negligible advantage in either Game 1 or Game 2.

Remark 3. We require the adversary to reveal its secret key to the challenger, because the challenger cannot, *in general*, otherwise be able to decrypt. By placing *constraints* on the PKE and IBE schemes used, it may be possible to eliminate this requirement.

Remark 4. In the sequel, we narrow our focus to pairing-based CBE schemes. In this context (and in the random oracle model), we construct CBE schemes secure against adaptive chosen ciphertext attack, where the parties do *not* need to reveal their secret keys.

3 A CBE Scheme Based on Boneh-Franklin

Roughly speaking, a CBE scheme is created by combining a PKE scheme and an IBE scheme. Since Boneh and Franklin's IBE scheme [6] is currently the most practical, we explicitly describe a CBE scheme that uses it. This scheme adds little online overhead to Boneh-Franklin: encryption complexity is increased slightly, but decryption complexity and ciphertext length are the same as in Boneh-Franklin.

3.1 Review of Pairings

Boneh-Franklin uses a bilinear map called a "pairing." Typically, the pairing used is a modified Weil or Tate pairing on a supersingular elliptic curve or abelian variety. However, we describe pairings and the related mathematics in a more general format here.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of some large prime order q. We write $\mathbb{G}_1$ additively and $\mathbb{G}_2$ multiplicatively.

Admissible pairings: We will call $\hat{e}$ an *admissible pairing* if $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a map with the following properties:

1. Bilinear: $\hat{e}(aQ, bR) = \hat{e}(Q, R)^{ab}$ for all $Q, R \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.
2. Non-degenerate: $\hat{e}(Q, R) \neq 1$ for some $Q, R \in \mathbb{G}_1$.
3. Computable: There is an efficient algorithm to compute $\hat{e}(Q, R)$ for any $Q, R \in \mathbb{G}_1$.

Notice that $\hat{e}$ is also symmetric – i.e., $\hat{e}(Q, R) = \hat{e}(R, Q)$ for all $Q, R \in \mathbb{G}_1$ – since $\hat{e}$ is bilinear and $\mathbb{G}_1$ is a cyclic group.

Bilinear Diffie-Hellman (BDH) Parameter Generator: As in [6], we say that a randomized algorithm $\mathcal{IG}$ is a BDH parameter generator if $\mathcal{IG}$ takes a security parameter $k > 0$, runs in time polynomial in k, and outputs the description of two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of the same prime order q and the description of an admissible pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

The security of the pairing-based schemes in this paper are based on the difficulty of the following problem:

BDH Problem: Given a randomly chosen $P \in \mathbb{G}_1$, as well as aP, bP, and cP (for unknown randomly chosen $a, b, c \in \mathbb{Z}/q\mathbb{Z}$), compute $\hat{e}(P, P)^{abc}$.

For the BDH problem to be hard, $\mathbb{G}_1$ and $\mathbb{G}_2$ must be chosen so that there is no known algorithm for efficiently solving the Diffie-Hellman problem in either $\mathbb{G}_1$ or $\mathbb{G}_2$. Note that if the BDH problem is hard for a pairing $\hat{e}$, then it follows that $\hat{e}$ is non-degenerate.

BDH Assumption: As in [6], if $\mathcal{IG}$ is a BDH parameter generator, the advantage $Adv_{\mathcal{IG}}(\mathcal{B})$ that an algorithm $\mathcal{B}$ has in solving the BDH problem is defined to be the probability that the algorithm $\mathcal{B}$ outputs $\hat{e}(P,P)^{abc}$ when the inputs to the algorithm are $\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP$ where $(G_1, \mathbb{G}_2, \hat{e})$ is $\mathcal{IG}$'s output for large enough security parameter k, P is a random generator of $\mathbb{G}_1$, and a, b, c are random elements of $\mathbb{Z}/q\mathbb{Z}$. The BDH assumption is that $Adv_{\mathcal{IG}}(\mathcal{B})$ is negligible for all efficient algorithms $\mathcal{B}$.

3.2 BasicCBE and FullCBE

As mentioned previously, IBE enables signatures to be used as decryption keys. For example, in Boneh-Franklin, a BLS signature [8] is used as a decryption key. Similarly, in "BasicCBE", we will use a two-signer BGLS aggregate signature [7] as a decryption key. (See [7] for details on the aggregate signature scheme.) Let k be the security parameter given to the setup algorithm, and let $\mathcal{IG}$ be a BDH parameter generator.

Setup: The CA:

1. runs $\mathcal{IG}$ on input k to generate groups $\mathbb{G}_1, \mathbb{G}_2$ of some prime order q and an admissible pairing $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$;
2. picks an arbitrary generator $P \in \mathbb{G}_1$;
3. picks a random secret $s_C \in \mathbb{Z}/q\mathbb{Z}$ and sets $Q = s_C P$;
4. chooses cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some n.

The system parameters are $params = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, H_1, H_2)$. The message space is $\mathcal{M} = \{0,1\}^n$. The CA's secret is $s_C \in \mathbb{Z}/q\mathbb{Z}$.

The CA uses its parameters and its secret to issue certificates. Assume that Bob's secret key / public key pair is $(s_B, s_B P)$, where $s_B P$ is computed according to the parameters issued by the CA. Bob obtains a certificate from his CA as follows.

Certification:

1. Bob sends $Bobsinfo$ to the CA, which includes his public key $s_B P$ and any necessary additional identifying information, such as his name.
2. The CA verifies Bob's information;
3. If satisfied, the CA computes $P_B = H_1(s_C P, i, Bobsinfo) \in \mathbb{G}_1$ in period i.
4. The CA then computes $Cert_B = s_C P_B$ and sends this certificate to Bob.

Before performing decryptions, Bob also signs $Bobsinfo$, producing $s_B P'_B$ where $P'_B = H_1(Bobsinfo)$. Now, notice that $S_{Bob} = s_C P_B + s_B P'_B$ is a two-person aggregate signature, as defined in [7]. Bob will use this aggregate signature as his decryption key!

Encryption: To encrypt $M \in \mathcal{M}$ using $Bobinfo$, Alice does the following:

1. Computes $P'_B = H_1(Bobsinfo) \in \mathbb{G}_1$.
2. Computes $P_B = H_1(Q, i, Bobsinfo) \in \mathbb{G}_1$.

3. Chooses a random $r \in \mathbb{Z}/q\mathbb{Z}$.
4. Sets the ciphertext to be:

$$C = [rP, M \oplus H_2(g^r)] \text{ where } g = \hat{e}(s_C P, P_B)\hat{e}(s_B P, P'_B) \in \mathbb{G}_2.$$

Notice that the length of the ciphertext is the same as in Boneh-Franklin. Alice can reduce her computation through precomputation – e.g., $\hat{e}(s_B P, P'_B)$ can likely be precomputed, since it is long-lived.

Decryption: To decrypt $[U, V]$, Bob computes:

$$M = V \oplus H_2(\hat{e}(U, S_{Bob})).$$

Notice that Bob's online decryption time is the same as in Boneh-Franklin.

BasicCBE is a one-way encryption scheme. It can be made secure (in the random oracle model) against adaptive chosen-ciphertext by, for example, using the Fujisaki-Okamoto transform in a manner similar to [6]. The transformed scheme, which we call "FullCBE", uses two additional cryptographic hash functions H_3 and H_4, and a semantically secure symmetric encryption scheme E:

Encryption: To encrypt $M \in \mathcal{M}$ using $Bobinfo$, Alice does the following:

1. Computes $P'_B = H_1(Bobsinfo) \in \mathbb{G}_1$.
2. Computes $P_B = H_1(Q, i, Bobsinfo) \in \mathbb{G}_1$.
3. Chooses random $\sigma \in \{0, 1\}^n$.
4. Sets $r = H_3(\sigma, M)$.
5. Sets the ciphertext to be:

$$C = [rP, \sigma \oplus H_2(g^r), E_{H_4(\sigma)}(M)] \text{ where } g = \hat{e}(s_C P, P_B)\hat{e}(s_B P, P'_B) \in \mathbb{G}_2.$$

Decryption: To decrypt $[U, V, W]$, Bob

1. Computes $\sigma = V \oplus H_2(\hat{e}(U, S_{Bob}))$.
2. Computes $M = E^{-1}_{H_4(\sigma)}(W)$.
3. Sets $r = H_3(\sigma, M)$ and rejects the ciphertext if $U \neq rP$.
4. Outputs M as the plaintext.

We can prove that FullCBE is secure in the random oracle model against adaptive chosen-ciphertext attack in two (quite different) ways – by showing such an attack implies 1) an existential forgery attack on the aggregate signature scheme, or 2) an adaptive chosen-ciphertext attack on BasicPub^{hy} (a public key encryption scheme defined in [6] and in Appendix A). We use the latter approach.

Lemma 1. *Let $\mathcal{A}$ be a IND-CCA adversary that has advantage ϵ against FullCBE. Suppose that $\mathcal{A}$ makes at most q_C certification queries and q_D decryption queries. Then, there is an IND-CCA adversary $\mathcal{B}$ with running time $O(time(A))$ that has advantage at least $\frac{\epsilon}{e(1+q_C+q_D)}$ against BasicPub^{hy}.*

Our proof, given in Appendix A, is similar to Boneh and Franklin's proof of Lemma 4.6 in [6]. Just like an IBE adversary is allowed to choose its identity adaptively, a CBE adversary is allowed to choose its information (including its public key) adaptively. When combined with Theorem 4.5 and Lemma 4.3 of [6], our Lemma 1 gives a reduction from the BDH problem to FullCBE, with time and advantage bounds as in Theorem 4.4 of [6].

4 Using Subset Covers to Reduce CA Computation

Using BasicCBE (or FullCBE), a CA that has N (currently valid) clients must compute N certificates per period. BasicCBE may therefore become impractical when N is large and updates are frequent – e.g., a CA with 225 million (currently valid) clients that performs hourly updates must compute 225 million certificates per hour (62500 per second) on average. In this section, we show how to reduce the CA's computation using *subset covers*.

4.1 The General Approach

Unlike in BasicCBE, we assume that the CA distributes a *long-lived* certificate to each client, which it periodically *reconfirms* (if appropriate). Before encrypting to Bob, Alice must obtain and verify Bob's long-lived certificate. This assumption costs us little, since Alice likely can obtain Bob's long-lived certificate when she obtains Bob's public key. (Recall that the main benefit of CBE is that Alice does not have to obtain *fresh* certificate status information; obtaining *long-lived* information is a simpler problem.) We will not concern ourselves with what certification scheme is used to generate the long-lived certificates, but we note that producing long-lived certificates should not add much to the CA's computational burden, since their production is amortized over a long time (assuming clients do not, say, sign-up "all at once.")

The CA arranges its $N < 2^m$ clients as leaves in an m-level binary tree by embedding a unique m-bit serial number (SN) in each client's long-lived certificate. For each time period, each tree node (including interior nodes) corresponds to an "identity," and the CA computes each node's decryption key according to an IBE scheme. Specifically, in time period i, the node corresponding to the $(k \leq m)$-bit SN $b_1 \cdots b_k$ may be mapped to $\langle i, b_1 \cdots b_k \rangle$. The node's associated decryption key is the IBE decryption key for $\langle i, b_1 \cdots b_k \rangle$.

We call $\langle i, b_1 \cdots b_k \rangle$ for $0 \leq k \leq m$ the *ancestors* of $\langle i, b_1 \cdots b_m \rangle$. In time period i, the CA finds a cover of the non-revoked clients – i.e., a set $\mathcal{S}$ of nodes, such that each of the $N - R$ non-revoked clients has an ancestor in $\mathcal{S}$, but none of the R revoked clients does. Such a cover, consisting of at most $R \log(N/R)$ nodes, can be found using the "Complete Subtree Method" described in [17]. The CA then publishes the decryption keys for each node in $\mathcal{S}$; we call these decryption keys *reconfirmation certificates*.

Now, assume that Alice has obtained and verified Bob's long-lived certificate, and therefore knows his public key and his SN. To encrypt to Bob, Alice does not need Bob's reconfirmation certificate. Instead, she encrypts her message $m + 1$ times, using (Bob's public key and) each of the $m+1$ identities of Bob's ancestors in the tree. If the CA has published a reconfirmation certificate corresponding to one of Bob's ancestors, Bob will be able to decrypt one of Alice's ciphertexts.

In summary, this scheme combines PKE and IBE just like BasicCBE, but it imposes a structure on the identities that permits efficient inclusion and exclusion (à la broadcast encryption). Though the complexity of encryption and the length of the ciphertext is increased, the CA computes only $R \log(N/R)$

reconfirmation certificates, rather than $N - R$ (in BasicCBE). Assuming that long-lived certificates expire 1 year after creation and that the yearly revocation rate is 10% (i.e., $R = .1N$) as in [15] and [16], the CA's computation is reduced by a factor of about 3.

4.2 A Pairing-Based CBE Scheme Using Subset Covers

The general approach above, while simple, creates difficulties in proving security against chosen-ciphertext attacks, since the same message is being encrypted under different keys. Below, we briefly present a variant of BasicCBE using subset covers, which deviates slightly from the general approach. As above, we assume that Bob has already obtained his long-lived certificate, which contains the serial number $b_1 \cdots b_m$.

Certification: The CA sets up as in BasicCBE and also uses $H_5 : \{0,1\}^* \rightarrow \mathbb{G}_1$ to map time periods to points. At the start of period i, the CA chooses random $x \in \mathbb{Z}/q\mathbb{Z}$ and finds a cover $\mathcal{S}$ of the non-revoked clients (using the Complete Subtree Method). Bob's reconfirmation certificate (if it exists) has the form $S_i = s_C T_i + xP_k$ together with xP, where $T_i = H_5(Q, i)$, $P_k = H_1(b_1 \cdots b_k)$ and $b_1 \cdots b_k \in \mathcal{S}$ is an ancestor of $b_1 \cdots b_m$.

Encryption: Alice has already verified Bob's initial certificate, and therefore knows $Bobsinfo$. Alice chooses random $r \in \mathbb{Z}/q\mathbb{Z}$. She sends the ciphertext $C = [rP, rP_1, \ldots, rP_m, V]$, where $V = M \oplus H_2(g^r)$ and $g = \hat{e}(Q, T_i)\hat{e}(s_B P, P'_B)$.

Decryption: Bob computes $M = V \oplus H_2(\frac{\hat{e}(rP, S_i + s_B P'_B)}{\hat{e}(xP, rP_k)})$.

Encryption involves $m+1$ point multiplications, but decryption involves only two pairing computations. Bob's reconfirmation certificate is concise: just two elements of $\mathbb{G}_1$. (Actually, since xP is common to all clients, we may say Bob's proof consists of just one element of G_1.) Elements of G_1 may be quite short – e.g., [8] proposes using an elliptic curve over $\mathbb{F}_{3^{97}}$ (about 154 bits) for the BLS signature scheme.

We note again that this scheme assumes that the client has already obtained a long-lived certificate. Thus, a Game 1 adversary cannot choose its public key with complete freedom; it must choose one of the N keys initially certified (and collude with the secret key holder). As with BasicCBE, the Fujisaki-Okamoto transform can be used to achieve CCA2 security. We remark, however, that gCCA2 security [2] may be considered preferable, because CCA2 security would technically require Bob to confirm that $rP_i = U_i$ for every i, even when $k \ll m$. We discuss the chosen-ciphertext security of this scheme in detail in the full version of the paper.

5 Incremental CBE Using Subset Covers

Intuitively, it seems like the CA's computation should be less for periods when few new revocations have occurred. However, this is not the case with the scheme of Section 4: if updates are hourly, the CA must compute about $R \log(N/R)$

reconfirmation certificates per hour (where R is the *total* number of revoked clients), even for hours when no client's certificate status has changed. In this section, we describe how to achieve "incremental CBE," where the CA's hourly computation is roughly proportional to the number of revocations that occurred during that hour. Specifically, the CA computes at most $R_{hour} \log(N/R_{hour})$ certificates per hour, where R_{hour} is the number of revocations during the previous hour. Assuming a 10% yearly revocation rate, a CA with 250 million clients performing hourly updates only needs to compute about 13 reconfirmation certificates per second, a *dramatic* improvement over BasicCBE. (We discuss incremental CBE's performance characteristics in more detail in Section 5.3.)

5.1 Basic Incremental CBE

As will become clearer from the detailed description below, this scheme is essentially based on two insights. First, as noted above and in [1], the CA can dramatically reduce its periodic computation by only revoking those clients that have become invalid in the past period. This strategy, however, has a price: a client cannot be considered currently certified (and must not be able to decrypt) unless it has an *unbroken chain* of periodic reconfirmation certificates for every period from the creation of its long-lived certificate to the present. At first, this seems to suggest that encryption and decryption complexity must be (at least) proportional to the number of time periods that have passed. Fortunately, using pairings, this is not the case: the second insight is an efficient way for each client to *consolidate* its periodic certificates into a single decryption key (consisting of $\log N + 1$ elements of $\mathbb{G}_1$), such that the encryption and decryption complexity and the ciphertext length are about $\log N$ times that of BasicCBE, regardless of how may periods have transpired since the client's initial certification.

We assume Bob has obtained a long-lived certificate containing his m-bit serial number $b_1 \cdots b_m$ and the period t_0 in which he was initially certified.

Certification: The CA sets up as in BasicCBE and also uses $H_5 : \{0,1\}^* \rightarrow \mathbb{G}_1$ to map time periods to points. At the start of period i, the CA chooses random $x \in \mathbb{Z}/q\mathbb{Z}$ and finds a cover $\mathcal{S}$ of the clients not revoked during period $i-1$ (using the Complete Subtree Method). Bob's reconfirmation certificate (if it exists) has the form $S_i' = s_C(T_i - T_{i-1}) + xP_k$ together with xP, where $T_i = H_5(Q, i)$, $P_k = H_1(b_1 \cdots b_k)$ and $b_1 \cdots b_k \in \mathcal{S}$ is an ancestor of $b_1 \cdots b_m$.

Consolidation: If the CA has continually reconfirmed Bob's key from its initial certification to the start of period i, we want it to be the case that Bob can compute a consolidated certificate of the form

$$S_i = s_C(T_i - T_{t_0}) + x_{i,1}P_1 + \cdots + x_{i,m}P_m \quad \text{with} \quad Q_{i,j} = x_{i,j}P \ \text{ for } \ 1 \leq j \leq m ,$$

for some $x_{i,j} \in \mathbb{Z}/q\mathbb{Z}$, $1 \leq j \leq m$, where $P_j = H_1(b_1 \cdots b_j)$. Assume he has a consolidated certificate with the correct form at the start of period $i-1$:

$$S_{i-1} = s_C(T_{i-1} - T_{t_0}) + x_{i-1,1}P_1 + \cdots + x_{i-1,m}P_m \ \text{ with } \ Q_{i-1,j} = x_{i-1,j}P \ .$$

Upon receiving S'_i and xP, Bob computes his new consolidated certificate as follows: $S_i = S_{i-1} + S'_i$, $Q_{i,j} = Q_{i-1,j}$ for $j \neq k$, and $Q_{i,k} = Q_{i-1,k} + xP$.

Encryption: Alice has already verified Bob's initial certificate, and therefore knows $Bobsinfo$. Alice chooses random $r \in \mathbb{Z}/q\mathbb{Z}$. She sends the ciphertext $C = [rP, rP_1, \ldots, rP_m, V]$, where $V = M \oplus H_2(g^r)$ and $g = \hat{e}(Q, T_i - T_{t_0})\hat{e}(s_B P, P'_B)$.

Decryption: Bob computes $M = V \oplus H_2(\frac{\hat{e}(rP, S_i + s_B P'_B)}{\prod_{j=1}^{m} \hat{e}(rP_j, Q_{i,j})})$.

Remark 5. If the CA gives Bob (say) $s_C T_{t_0} + x_{t_0} P_m$ with $Q_{t_0,m} = x_{t_0} P$ (for some $x_{t_0} \in \mathbb{Z}/q\mathbb{Z}$) at the time of initial certification, and Bob consolidates this with his periodic certificates, then "$-T_{t_0}$" need not be included in the Encryption step. However, the CA may prefer not to constrain how it handles initial certification.

5.2 Security

In the full version of the paper, we prove (in the random oracle model, under the Bilinear Diffie-Hellman Assumption, and using the Fujisaki-Okamoto transform) that Bob needs an unbroken chain of reconfirmation certificates – from the time of his initial certification to the present – to decrypt. We could prove the scheme secure against Game 2 adversaries using similar techniques.

Here, in lieu of proof, we provide some rough intuitive justification for incremental CBE's security. S_i contains a time component of the form $s_C(T_i - T_{t_0})$ and an identity component of the form $x_1P_1 + x_2P_2 + \cdots + x_mP_m$. If Bob tries to consolidate his certificates without his reconfirmation certificate for day z, $t_0 < z \leq i$, the result will have the form $s_C(T_i - T_z + T_{z-1} - T_{t_0}) + x_1P_1 + x_2P_2 + \cdots + x_mP_m$. In other words, the time component of S_i will have the incorrect form. If Bob tries to substitute someone else's day z reconfirmation point $s_C(T_z - T_{z-1}) + x'_k P'_k$, where P'_k does not correspond to one of Bob's ancestors, the result will have the form $s_C(T_i - T_{t_0}) + x_1P_1 + x_2P_2 + \cdots + x_mP_m + x'_kP'_k$. In other words, the identity component of S_i will have the incorrect form.

5.3 Performance Characteristics

Like the other CBE schemes, incremental CBE enjoys all the infrastructural benefits of eliminating third-party queries. Also, the CA's computation is minimal; it computes only 13 reconfirmation certificates per second. Since each reconfirmation certificate is essentially equivalent (computation-wise) to a BLS signature, which requires only 3.57 ms to compute on a Pentium III 1 GHz [3], a single PC can easily handle the computation.

Distributing the reconfirmation certificates can be handled in a variety of ways. Since CBE eliminates third-party queries, one interesting alternative to the usual directories-based approach is to "push" certificates directly to the clients they certify – i.e., the CA sends each client its reconfirmation certificate rather than waiting for the client's query. This would eliminate queries altogether.[6] If

[6] In practice, the CA would probably allow *some* queries, both by clients and third-parties. However, CBE allows a CA to discourage queries – e.g., by using fees – so that the number of queries is reduced to a desired level.

the CA uses *separate transmissions for each client*, the CA's required bandwidth is (320 bits per cert)$*$(225 million certs per hour)/(3600 seconds per hour) = 20 mbits/sec. This figure is a lower bound; it does not include overhead such as packet headers. Such a large transmission may need to be delegated to a number of servers.

Theoretically, however, if the CA pushes its certificates, it can dramatically reduce its bandwidth requirements by using *multicast*. Recall that a reconfirmation certificate for node $b_1 \cdots b_k$ applies to all of $b_1 \cdots b_k$'s descendants. So, if the CA sets up multicast address for each interior node, the CA's expected bandwidth requirements are very low – only about (160 bits per cert)$*$(13 certs per second) $\approx$ 2.1 kbits/sec. (The certificates are 160 bits in this case, since xP, the other 160-bit value, can be sent to all clients via one multicast). Even if the CA sets up multicast addresses only for high-level nodes (close to the root), it can significantly reduce its bandwidth requirements.

For users, incremental CBE is somewhat expensive computationally: encryption and decryption cost about $m = \lceil \log N \rceil$ times that of Boneh-Franklin. This may make the scheme impractical for some applications in the very near future. It may not be a problem for other applications, like email, where fast online computation is not such a concern. We expect that, in the future, computational considerations will become less important relative to network considerations (like the latency caused by the (fixed) speed of light), making CBE's advantages more prominent. If desired, however, one can shift some user computation back to the CA by "fattening" the binary tree (so that each node has more children), or having the CA maintain several trees concurrently, as described in [1].

The total length of a client's consolidated certificate, which can be used as explicit proof of certification (even of signature keys), never grows beyond $160(m+1)$ bits. Even though CBE was not originally intended to improve the efficiency of explicit certification, we note that explicit proofs in incremental CBE are more compact than in Aiello-Lodha-Ostrovsky's incremental scheme [1], where the length grows linearly with the number of time periods.

6 Extensions and Generalizations

6.1 High-Granularity CBE

Suppose a one hour time granularity is insufficient; we want certificate revocation to be practically instantaneous. One option is for Alice to encrypt using $i+1$ as the time period during period i. The drawback is that Bob, after receiving Alice's message, may need to wait an hour to decrypt it. Another option, called the SEM architecture, is described in [5]. In SEM, revocation is instantaneous, and so is Bob's ability to decrypt, but he must interact with a "security mediator" for each message decryption. Below, we briefly describe a "high-granularity" version of CBE, where revocation is practically instantaneous – say, within 1 second – but where Bob's interaction with the CA grows not with the number of messages he decrypts, but rather with the number of his "sessions."

In our scheme, the CA uses a recent forward-secure encryption (FSE) scheme [9], but *in reverse* – i.e., the CA relabels the time periods such that given the FSE decryption key for period i, one can compute the decryption key for period j if $j < i$, but not if $j > i$.[7] Bob may download such a certificate a few times daily, depending on how often he checks his messages, and use it to decrypt messages from multiple previous time periods. As in [9], the size of this certificate is merely logarithmic in the number of periods for which messages can be decrypted; thus, we can make revocation highly granular without sacrificing much efficiency. Also, this scheme can be combined with (say) incremental CBE: the CA performs hourly updates as in incremental CBE, but employs a subtree of 3600 seconds for clients that request high-granularity certificates.

6.2 Hierarchical CBE

Though our previous schemes have used only one CA, adapting CBE to a hierarchy of CAs is fairly straightforward. Perhaps the more obvious approach is simply to combine a HIBE scheme and a PKE scheme, much as CBE combines IBE and PKE. However, using this approach, encryption and decryption complexity, as well as ciphertext length, are all about t times that of Boneh-Franklin, where t is the level of the recipient in the hierarchy. Instead, we use the BGLS aggregate signature scheme. The encryption complexity of this scheme is still proportional to t, but the decryption complexity and ciphertext length are identical to Boneh-Franklin. Suppose Bob is at level t, that his public key is $s_t P$, and that the CAs above him have public keys $s_j P$ for $0 \leq j \leq t-1$.

Certification of CAs: CA_j certifies $s_{j+1}P$ as CA_{j+1}'s public key by producing a signature of the form $s_j P_{j+1}$, where $P_{j+1} = H_1(s_j P, CA_{j+1}info)$ and $CA_{j+1}info$ includes $s_{j+1}P$.

Certification of Bob: Similarly, Bob's parent CA produces a certificate of the form $s_{t-1}P_t$, where $P_t = H_1(s_{t-1}P, Bobsinfo)$ and $Bobsinfo$ includes $s_t P$.

Aggregation: Bob signs his key to produce $s_t P'_B$ and "aggregates" this signature with the certificates in his chain simply by adding them together: $S_{Agg} = s_t P'_B + \sum_{j=1}^{t} s_{j-1} P_j$.

Encryption: We assume Alice knows $Bobsinfo$ and $CA_j info$ for $0 \leq j \leq t-1$. Alice chooses random $r \in \mathbb{Z}/q\mathbb{Z}$ and sends $C = [rP, V]$, where $V = M \oplus H_2(g^r)$ and $g = \hat{e}(P'_B, s_t P) \prod_{j=1}^{t} \hat{e}(P_j, s_{j-1}P)$.

Decryption: Bob computes $M = V \oplus H_2(\hat{e}(rP, S_{Agg}))$.

Remark 6. To handle revocation, we can embed time periods in the various certificates above. However, Alice must know the certification "schedules" of all of Bob's ancestral CAs, which may make implementation difficult in practice.

Remark 7. We note that the second scheme is useful outside of the PKI setting. It provides a general way for making a keyholder's decryption ability contingent on that keyholder's acquisition of multiple signatures / authorizations.

[7] We omit the details of our scheme, since we would largely be rehashing [9].

6.3 Other Generalizations

It is interesting that even "exotic" pairing-based signatures can typically be used as decryption keys. For example, Alice can make Bob's ability to decrypt contingent on his possession of a BGLS ring signature [7].

Ring Signing: The signer is given public keys $\{Q_1 = s_1P, \ldots, Q_n = s_nP\}$, s_k for some $1 \le k \le n$, and message M'. It computes $P_{M'} = H_1(M')$, chooses random $a_i \in \mathbb{Z}/q\mathbb{Z}$ for $i \neq k$, and sends $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ to Bob, where $\sigma_k = s_k^{-1}P_{M'} - \sum_{i \neq k} a_iQ_i$ and $\sigma_i = a_iQ_k$ for $i \neq k$.

Ring Encryption: We assume that Alice knows Bob's public key s_BP, the public keys $\{Q_1, \ldots, Q_n\}$ and the message M'. Alice chooses random $r \in \mathbb{Z}/q\mathbb{Z}$ and sends $C = [rP, rQ_1, \ldots, rQ_n, M \oplus H_2(g^r)]$, where $g = \hat{e}(P, P_{M'})\hat{e}(s_BP, P'_B)$.

Ring Decryption: Given the ciphertext $[U, U_1, \ldots, U_n, V]$, Bob computes $M = V \oplus H_2(h)$, where $h = \hat{e}(U, s_BP'_B)\prod_{i=1}^{n} \hat{e}(U_i, \sigma_i)$.

7 Summary

We described the notion of certificate-based encryption, and demonstrated how it streamlines PKI. The key idea is that certificate-based encryption enables implicit certification without the problems of IBE, and that implicit certification allows us to eliminate third-party queries on certificate status, thereby reducing infrastructural requirements. We also described an incremental CBE scheme that reduces the CA's computation and bandwidth requirements to exceptionally low levels, even though the scheme does not use hash chains or trees like previous PKI proposals.

References

1. W. Aiello, S. Lodha, and R. Ostrovsky. Fast Digital Identity Revocation. In *Proc. of Crypto 1998*, LNCS 1462, pages 137–152. Springer-Verlag, 1998.
2. J.H. An, Y. Dodis and T. Rabin. On the Security of Joint Signature and Encryption. In *Proc. of Eurocrypt 2002*, LNCS 2332, pages 83–107. Springer-Verlag, 2002.
3. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Proc. of Crypto 2002*, LNCS 2442, pages 354–368. Springer-Verlag, 2002.
4. M. Bellare and A. Palacio. Protecting against Key Exposure: Strongly Key-Insulated Encryption with Optimal Threshold. Available at http://eprint.iacr.org, 2002.
5. D. Boneh, X. Ding, G. Tsudik, M. Wong. A Method for Fast Revocation of Public Key Certificates and Security Capabilities. In *Proc. of 10th Annual USENIX Security Symposium*, 2001, available at http://crypto.stanford.edu/˜dabo/pubs.html.
6. D. Boneh and M. Franklin. Identity-Based Encryption from the Weil pairing. In *Proc. of Crypto 2001*, LNCS 2139, pages 213–229. Springer-Verlag, 2001.
7. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Proc. of Eurocrypt 2003* (to appear).

8. D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Proc. of Asiacrypt 2001*, LNCS 2248, pages 514–532. Springer-Verlag, 2001.
9. R. Canetti, S. Halevi, J. Katz. A Forward-Secure Public-Key Encryption Scheme. In *Proc. of Eurocrypt 2003* (to appear).
10. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-Insulated Public Key Cryptosystems. In *Proc. of Eurocrypt 2002*, LNCS 2332, pages 65–82. Springer-Verlag, 2002.
11. E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Proc. of Crypto 1999*, LNCS 1666, pages 537–554. Springer-Verlag, 1999.
12. I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and Fresh Certification. In *Proc. of Public Key Cryptography 2000*, LNCS 1751, pages 342–353. Springer-Verlag, 2000.
13. C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptography. In *Proc. of Asiacrypt 2002*, LNCS 2501, pages 548-566. Springer-Verlag, 2002.
14. S. Micali. Efficient Certificate Revocation. Technical Report TM-542b, MIT Laboratory for Computer Science, 1996.
15. S. Micali. Novomodo: Scalable Certificate Validation and Simplified PKI Management. In *Proc. of 1st Annual PKI Research Workshop*, 2002, available at http://www.cs.dartmouth.edu/~pki02/.
16. M. Naor and K. Nissim. Certificate Revocation and Certificate Update. In *Proc. of 7th Annual USENIX Security Symposium*, 1998, available at http://www.wisdom.weizmann.ac.il/~kobbi/papers.html.
17. D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In *Proc. of Crypto 2001*, LNCS 2139, pages 41-62. Springer-Verlag 2001.
18. A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Proc. of Crypto 1984*, LNCS 196, pages 47–53. Springer-Verlag, 1985.

A Proof of Lemma 1

First, we describe BasicPub^{hy}, a public key cryptosystem that uses the Fujisaki-Okamoto transform to achieve chosen-ciphertext security under the BDH Assumption. Let k be the security parameter given to the setup algorithm, and let $\mathcal{IG}$ be a BDH parameter generator.

Setup: The keyholder:

1. runs $\mathcal{IG}$ on input k to generate groups $\mathbb{G}_1, \mathbb{G}_2$ of some prime order q and an admissible pairing $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$;
2. picks arbitrary generators $P, P_1 \in \mathbb{G}_1$;
3. picks a random secret $s_C \in \mathbb{Z}/q\mathbb{Z}$ and sets $Q = s_C P$;
4. chooses cryptographic hash functions $H_2 : \mathbb{G}_2 \to \{0,1\}^n$.
5. uses hash functions H_3 and H_4 and semantically secure encryption scheme E, as specified in Fujisaki-Okamoto.

The public key is $K_{pub} = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_1, Q, H_2, H_3, H_4, E)$, and the secret key is $s_C P_1$. The message space is $\mathcal{M} = \{0,1\}^n$.

Encryption: To encrypt $M \in \mathcal{M}$, the encrypter:

1. Chooses random $\sigma \in \{0,1\}^n$.

2. Sets $r = H_3(\sigma, M)$.
3. Sets the ciphertext to be:

$$C = [rP, \sigma \oplus H_2(g^r), E_{H_4(\sigma)}(M)] \text{ where } g = \hat{e}(Q, P_1).$$

Decryption: To decrypt $[U, V, W]$, Bob

1. Computes $\sigma = V \oplus H_2(\hat{e}(U, s_C P_1))$.
2. Computes $M = E^{-1}_{H_4(\sigma)}(W)$.
3. Sets $r = H_3(\sigma, M)$ and rejects the ciphertext if $U \neq rP$.
4. Outputs M as the plaintext.

Now, we show how to construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage $\epsilon/e(1+q_C+q_D)$ against BasicPub^{hy}. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running the key generation algorithm of BasicPub^{hy}. The result is a public key $K_{pub} = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_1, Q, H_2, H_3, H_4, E)$ and a private key $s_C P_1$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ have order q and $Q = s_C P$ for $s_C \in \mathbb{Z}/q\mathbb{Z}$. The challenger gives K_{pub} to $\mathcal{B}$. Now, $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

Setup: $\mathcal{B}$ gives $\mathcal{A}$ the FullCBE $params = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, Q, H_1, H_2, H_3, H_4, E)$, where H_1 is a random oracle controlled by $\mathcal{B}$.

H_1**-queries:** $\mathcal{A}$ can make an H_1-query at any time. There are two types of H_1-queries. A Type-1 query (to compute P_B in FullCBE) is of the form $(i_j, s_j P, w_j)$ – where one may view i_j as a time period, $s_j P$ as the public key that $\mathcal{A}$ wants certified, and w_j as other information that may be in a certificate (such as a name). A Type-2 query (to compute P'_B in FullCBE) is of the form $(s_j P, w_j)$. The query is parsed before processing. For consistency, $\mathcal{B}$ maintains an H_1-list logging its H_1-query responses. This list is initially empty. $\mathcal{B}$ responds to $\mathcal{A}$'s H_1-query as follows:

1. If $\mathcal{A}$ made the same H_1-query previously, as indicated by the H_1-list, $\mathcal{B}$ responds the same way as it did before.
2. If it is a Type-2 query:
   (a) $\mathcal{B}$ generates random $b_j \in \mathbb{Z}/q\mathbb{Z}$ and sets $P'_j = b_j Q$.
   (b) $\mathcal{B}$ adds tuple $(s_j P, w_j, b_j)$ to the H_1-list, and returns P'_j to $\mathcal{A}$.
3. If it is a Type-1 query:
   (a) $\mathcal{B}$ runs a Type-2 query on $(s_j P, w_j)$ to recover b_j.
   (b) $\mathcal{B}$ generates a random $coin_j \in \{0,1\}$ so that Pr$[coin = 0]= \delta$ for δ to be determined later.
   (c) $\mathcal{B}$ generates random $c_j \in \mathbb{Z}/q\mathbb{Z}$. If $coin_j = 0$, it sets $P_j = c_j P$; else, it sets $P_j = c_j P_1 - b_j s_j P$.
   (d) $\mathcal{B}$ adds tuple $(i_j, s_j P, w_j, c_j, coin_j)$ to the H_1-list, and returns P_j to $\mathcal{A}$.

Note that P'_j and P_j are uniform in $\mathbb{G}_1$ and independent of $\mathcal{A}$'s view as required.

Phase 1 – Certification Queries: $\mathcal{B}$ responds to $\mathcal{A}$'s certification query $(i_j, w_j, s_j P)$ as follows:

1. $\mathcal{B}$ runs the H_1-query response algorithm on (i_j, w_j, s_jP) to recover c_j and $coin_j$. If $coin_j = 1$, $\mathcal{B}$ terminates. The attack on BasicPub^{hy} failed.
2. Otherwise, $coin_j = 0$. $\mathcal{B}$ gives $\mathcal{A}$ its certificate: $s_CP_j = c_jQ$.

Phase 1 – Decryption Queries: Let (i_j, s_jP, w_j, C_j) be a decryption query issued by $\mathcal{A}$, where $C_j = (U_j, V_j, W_j)$. $\mathcal{B}$ responds as follows.

1. $\mathcal{B}$ runs the H_1-query response algorithm on (i_j, w_j, s_jP) to recover b_j, c_j and $coin_j$.
2. Suppose $coin_j = 0$. Then, $\mathcal{B}$ computes the decryption key $s_CP_j + s_jP'_j = c_jQ + b_j(s_jP)$, and uses it to decrypt C_j.
3. Suppose $coin_j = 1$. Then, $\mathcal{B}$ sets $C'_j = (c_jU, V, W)$. $\mathcal{B}$ relays C'_j to the challenger, and relays the challenger's response back to $\mathcal{A}$.

Recall that the challenger's private key is s_CP_1. Now, notice that $\hat{e}(c_jU, s_CP_1) = \hat{e}(rP, c_js_CP_1) = \hat{e}(rP, s_C(c_jP_1 - b_js_jP) + s_jb_jQ) = \hat{e}(rP, s_CP_j + s_jP'_j)$, where $s_CP_j + s_jP'_j$ is the FullCBE decryption key. Thus, the challenger provides the correct decryption of C_j.

Challenge: Once $\mathcal{A}$ decides that Phase 1 is over, it requests a challenge ciphertext on (i_z, s_zP, w_z) for message M_0 or M_1. $\mathcal{B}$ responds as follows:

1. $\mathcal{B}$ relays M_0 and M_1 to the challenger as the messages that it wants to be challenged on. The challenger sends $\mathcal{B}$ a BasicPub^{hy} ciphertext $C = [U, V, W]$ such that C is an encryption M_x, $x \in \{0, 1\}$.
2. $\mathcal{B}$ runs the H_1-query response algorithm on (i_z, s_zP, w_z) to recover c_z and $coin_z$. If $coin_z = 0$, $\mathcal{B}$ terminates. The attack on BasicPub^{hy} failed.
3. Otherwise, $coin_z = 1$. $\mathcal{B}$ gives $\mathcal{A}$ its challenge ciphertext: $C' = [c_j^{-1}U, V, W]$.

Notice that $\hat{e}(c_j^{-1}U, s_CP_j + s_jP'_j) = \hat{e}(c_j^{-1}U, c_js_CP_1) = \hat{e}(U, s_CP_1)$, where s_CP_1 it the challenger's decryption key. Thus, the challenge ciphertext is an encryption of M_x in FullCBE as required.

Phase 2 – Certification Queries: $\mathcal{B}$ responds as in Phase 1.

Phase 2 – Decryption Queries: $\mathcal{B}$ responds as in Phase 1, except it terminates if the decryption query to be relayed to the challenger is equal to C.

Guess: $\mathcal{A}$ guesses x' for x. $\mathcal{B}$ also uses x' as its guess.

Claim: If algorithm $\mathcal{B}$ does not abort during the simulation, then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, if $\mathcal{B}$ does not abort then $\Pr[M = M'] \geq \epsilon$. The probability is over the random bits used by $\mathcal{A}$, $\mathcal{B}$ and the challenger.

Proof of Claim: All responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1$. All responses to certification and decryption queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the FullCBE encryption of M_x under the the public key (and other information) chosen by $\mathcal{A}$. Therefore, by the definition of algorithm $\mathcal{A}$, it will output $x' = x$ with probability at least ϵ.

Probability: It remains to calculate the probability that $\mathcal{B}$ aborts during simulation. This analysis is identical to that in [6], and is therefore omitted.

Remark 8. The above proves FullCBE secure (in the random oracle model, under the BDH Assumption) against an adaptive chosen ciphertext attack by a Game 1 adversary. The proof for Game 2 adversaries is similar and therefore omitted.

Remark 9. Recall that, in Section 2, we required a Game 1 adversary to reveal its personal secret key in its queries. However, this does not occur above; rather than revealing s_j, $\mathcal{A}$ "proves its knowledge" of s_j through its ability to compute $s_j P'_j$. Since the adversary does not need to reveal its secret key, the proof above is slightly *stronger* than required. That is, BasicCBE conforms to Section 2's security model, even though clients do not actually reveal their personal secret keys to the CA, which, indeed, would defeat the purpose of CBE.