Certificateless Public Key Cryptography

Sattam S. Al-Riyami and Kenneth G. Paterson*
Information Security Group,
Royal Holloway, University of London, Egham, Surrey, TW20 0EX
s.al-riyami@rhul.ac.uk kenny.paterson@rhul.ac.uk

Abstract

This paper introduces and makes concrete the concept of *certificateless public key cryptography* (CL-PKC), a model for the use of public key cryptography which avoids the inherent escrow of identity-based cryptography and yet which does not require certificates to guarantee the authenticity of public keys. The lack of certificates and the presence of an adversary who has access to a master key necessitates the careful development of a new security model. We focus on certificateless public key encryption (CL-PKE), showing that a concrete pairing-based CL-PKE scheme is secure provided that an underlying problem closely related to the Bilinear Diffie-Hellman Problem is hard.

1 Introduction

The main difficulty today in developing secure systems based on public key cryptography is not the problem of choosing appropriately secure algorithms or implementing those algorithms. Rather, it is the deployment and management of infrastructures to support the authenticity of cryptographic keys: there is a need to provide an assurance to the user about the relationship between a public key and the identity (or authority) of the holder of the corresponding private key. In a traditional Public Key Infrastructure (PKI), this assurance is delivered in the form of certificate, essentially a signature by a Certification Authority (CA) on a public key [1]. The problems of PKI technology are well documented, see for example [16]. Of note are the issues associated with certificate management, including revocation, storage and distribution and the computational cost of certificate verification. These are particularly acute in processor or bandwidth-limited environments [9].

Identity-based public key cryptography (ID-PKC), first proposed by Shamir [22], tackles the problem of authenticity of keys in a different way to traditional PKI. In ID-PKC, an entity's public key is derived directly from certain aspects of its identity. Private keys are generated for entities by a trusted third party called a private key generator (PKG). The first fully practical and secure identity-based public key encryption scheme was presented in [5]. Since then, a rapid development of ID-PKC has taken place, see [18] for a brief survey. It has also

*This author supported by the Nuffield Foundation, NUF-NAL 02.

been illustrated in [8, 18, 24] how ID-PKC can be used as a tool to enforce what might be termed "cryptographic work-flows", that is, sequences of operations (e.g. authentications) that need to be performed by an entity in order to achieve a certain goal.

The direct derivation of public keys in ID-PKC eliminates the need for certificates and some of the problems associated with them. On the other hand, the dependence on a PKG who uses a system-wide master key to generate private keys inevitably introduces key escrow to ID-PKC systems. For example, the PKG can decrypt any ciphertext in an identity-based public key encryption scheme. Equally problematical, the PKG could forge any entity's signatures in an identity-based signature scheme, so ID-PKC cannot offer true non-repudiation in the way that traditional PKI can. The escrow problem can be solved to a certain extent by the introduction of multiple PKGs and the use of threshold techniques, but this necessarily involves extra communication and infrastructure. Moreover, the compromise of the PKG's master key could be disastrous in an ID-PKC system, and usually more severe than the compromise of a CA's signing key in a traditional PKI. For these reasons, it seems that the use of ID-PKC may be restricted to small, closed groups or to applications with limited security requirements.

1.1 Certificateless Public Key Cryptography

In this paper, we introduce a new paradigm for public key cryptography, which we name certificateless public key cryptography (CL-PKC). Our concept grew out of a search for public key schemes that do not require the use of certificates and yet do not have the built-in key escrow feature of ID-PKC. The solution we propose enjoys both of these properties; it is a model for the use of public key cryptography that is intermediate between traditional PKI and ID-PKC.

We demonstrate that our concept of CL-PKC can be made real by specifying certificateless encryption and signature schemes. We prove that the encryption scheme is secure in a new and appropriate model, given the hardness of an underlying computational problem. Further development of our concept and more certificateless schemes can be found in the full version of this paper, [2].

1.2 Defining CL-PKC

We sketch the defining characteristics of CL-PKC.

A CL-PKC system still makes use of TTP which we name the key generating centre (KGC). By way of contrast to the PKG in ID-PKC, this KGC does *not* have access to entities' private keys. Instead, the KGC supplies an entity A with a *partial private key* D_A which the KGC computes from an identifer $\mathtt{ID}_A$ for the entity and a master key. Note that we will often equate A with its identifier $\mathtt{ID}_A$. The process of supplying partial private keys should take place confidentially and authentically: the KGC must ensure that the partial private keys are delivered securely to the correct entities. Identifiers can be arbitrary strings.

The entity A then combines its partial private key D_A with some secret information to generate its actual private key S_A. In this way, A's private key is not available to the KGC. The entity A also combines its secret information with the KGC's public parameters to compute its public key P_A. Note that A need not be in possession of S_A before generating P_A: all that is needed to generate both is the same secret information. The system is not identity-based, because the public key is no longer computable from an identity (or identifier) alone.

Entity A's public key might be made available to other entities by transmitting it along with messages (for example, in a signing application) or by placing it in a public directory (this would be more appropriate for an encryption setting). But no further security is applied to the protection of A's public key. In particular, there is no certificate for A's key. To encrypt a message to A or verify a signature from A, entity B makes use of P_A and $\mathtt{ID}_A$.

A more formal model for certificateless public key encryption (CL-PKE) will be given in Section 3. Much of this model is also applicable for our other certificateless primitives.

1.3 An Adversarial Model for CL-PKC

Because of the lack of authenticating information for public keys (in the form of a certificate, for example), we must assume that an adversary can replace A's public key by a false key of its choice. This might seem to give the adversary tremendous power and to be disastrous for CL-PKC. However, we will see that an active adversary who attacks our concrete schemes in this way gains nothing useful: without the correct private key, whose production requires the partial private key and therefore the cooperation of the KGC, an adversary will not be able to decrypt ciphertexts encrypted under the false public key, produce signatures that verify with the false public key, and so on.

Of course, we must assume that the KGC does not mount an attack of this type: armed with the partial private key and the ability to replace public keys, the KGC could impersonate any entity in generating a private/public key pair and then making the public key available. Thus we must assume that, while the KGC is in possession of the master key and hence all partial private keys, it is trusted not to replace entities' public keys. However, we assume that the KGC might engage in other adversarial activity, eavesdropping on ciphertexts and making decryption queries, for example. In this way, users invest roughly the same level of trust in the KGC as they would in a CA in a traditional PKI – it is rarely made explicit, but such a CA is always assumed not to issue new certificates binding arbitrary public keys and entity combinations of its choice, and especially not for those where it knows the corresponding private key! When compared to ID-PKC, the trust assumptions made of the trusted third party in CL-PKC are much reduced: in ID-PKC, users must trust the PKG not to abuse its knowledge of private keys in performing passive attacks, while in CL-PKC, users need only trust the KGC not to actively propagate false public keys.

The word *roughly* here merits further explanation. In a traditional PKI, if the CA forges certificates, then the CA can be identified as having misbehaved through the existence of two valid certificates for the same identity. This is not the case in our schemes: a new public key could have been created by the legitimate user or by the KGC, and it cannot be easily decided which is the case. The terminology of [15] is useful here: our schemes achieve trust level 2, whereas a traditional PKI reaches trust level 3. However, we can further strengthen security against a malicious KGC in our schemes by allowing entities to bind together their public keys and identities. Now the existence of two different, working public keys for the same identity will identify the KGC as having misbehaved in issuing both corresponding partial private keys. Details of this modification can be found in Section 5.1. With this binding in place, our schemes do reach trust level 3.

In Section 3, we will present an adversarial model for CL-PKE which captures these capabilities in a formal way. The model we present there is a natural generalization of the fully adaptive, multi-user model of [5] to the CL-PKC setting, and involves two distinct types of adversary: one who can replace public keys at will and another who has knowledge of the master key but does not replace public keys. Given our detailed development of this model, the adaptations to existing models that are needed to produce adversarial models for other certificateless primitives become straightforward.

1.4 Implementation and Applications of CL-PKC

Our presentation of CL-PKC schemes will be at a fairly abstract level, in terms of bilinear maps on groups. However, the concrete realization of these schemes using pairings on elliptic curves is now becoming comparatively routine, after the work of [3, 6, 7, 12] on implementation of pairings and selection of curves with suitable properties. All the schemes we present use a small number of pairing calculations for each cryptographic operation, and some of these can usually be eliminated when repeated operations involving the same identities take place. Public and private keys are small in size: two elliptic curve points for the public key and one for the private key.

The infrastructure needed to support CL-PKC is lightweight when compared to a traditional PKI. This is because, just as with ID-PKC, the need to manage certificates is completely eliminated. This immediately makes CL-PKC attractive for low-bandwidth, low-power situations. However, it should be pointed out that recently introduced signatures schemes enjoying very short signatures [7] could be used to significantly decrease the size of certificates and create a lightweight PKI. Our CL-PKC signature scheme can also support true non-repudiation, because private keys remain in the sole possession of their legitimate owners.

Revocation of keys in CL-PKC systems can be handled in the same way as in ID-PKC systems. In [5] the idea of appending validity periods to identifiers $\mathtt{ID}_A$ is given as one convenient solution. In the context of CL-PKC, this ensures that any partial private key, and hence any private key, has a limited shelf-life.

As will become apparent, our CL-PKC schemes are actually very closely related to existing pairing-based ID-PKC schemes. One consequence of this is that any infrastructure deployed to support pairing-based ID-PKC (e.g. a PKG) can also be used to support our CL-PKC schemes too: in short, the two types of scheme can peacefully co-exist. In fact, an entity can be granted a private key for a pairing-based ID-PKC scheme and immediately convert it into a private key for our CL-PKC scheme. In this way, an entity who wishes to prevent the PKG exploiting the escrow property of an identity-based system can do so, though at the cost of losing the identity-based nature of its public key.

Although our CL-PKC schemes are no longer identity-based, they do enjoy the property that an entity's private key can be determined after its public key has been generated and used. This is a useful feature. An entity B can encrypt a message for A using A's chosen public key and an identifier $\mathtt{ID}_A$ of B's choice. This identifier should contain A's identity but might also contain a condition that A must demonstrate that it satisfies before the KGC will deliver the corresponding partial private key (which in turn allows A to compute the right private key for decryption). For more applications of "cryptographic workflows" which cannot be supported using certificate-based systems, see [18, 24].

1.5 Related Work

Our work on CL-PKC owes much to the pioneering work of Boneh and Franklin [5, 6] on identity-based public key encryption. In fact, our CL-PKE scheme is derived from the scheme of [5] by making a very simple modification (albeit, one with far-reaching consequences). Our security proofs require significant changes and new ideas to handle our new types of adversary. Likewise, our signature and other schemes [2] also arise by adapting existing ID-PKC schemes. Another alternative to traditional certificate-based PKI called self-certified keys was introduced by Girault [15] and further developed in [19, 21]. The properties of the schemes presented in [15, 19, 21] are compared to CL-PKC in the full version [2].

Recent and independent work of Gentry [13] simplifies certificate management in traditional PKI systems in a very neat way by exploiting pairings. Gentry's scheme is presented in the context of a traditional PKI model, whereas our work departs from the traditional PKI and ID-PKC models to present a new paradigm for the use of public-key cryptography. Moreover, the concrete realizations of the two models are different. However, it is possible to re-cast Gentry's work to divorce it from the setting of a traditional PKI. Further discussion can be found in [2].

2 Background Definitions

Throughout the paper, $\mathbb{G}_1$ denotes an additive group of prime order q and $\mathbb{G}_2$ a multiplicative group of the same order. We let P denote a generator of $\mathbb{G}_1$. For us, a pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ with the following properties: (1) The map e is bilinear: given $Q, W, Z \in \mathbb{G}_1$, we have $e(Q, W + Z) = e(Q, W) \cdot e(Q, Z)$ and $e(Q + W, Z) = e(Q, Z) \cdot e(W, Z)$. (2) The map e is non-degenerate: $e(P, P) \neq 1_{\mathbb{G}_2}$. (3) The map e is efficiently computable.

Typically, the map e will be derived from either the Weil or Tate pairing on an elliptic curve over a finite field. We refer to [3, 6, 7, 12] for a more comprehensive description of how these groups, pairings and other parameters should be selected in practice for efficiency and security.

We also introduce here the computational problems that will form the basis of security for our CL-PKC schemes.

Bilinear Diffie-Hellman Problem (BDHP): Let $\mathbb{G}_1$, $\mathbb{G}_2$, P and e be as above. The BDHP in $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$ is as follows: Given $\langle P, aP, bP, cP \rangle$ with uniformly random choices of $a, b, c \in \mathbb{Z}_q^*$, compute $e(P, P)^{abc} \in \mathbb{G}_2$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the BDHP in $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$ if
$\Pr\left[\mathcal{A}(\langle P, aP, bP, cP \rangle) = e(P, P)^{abc}\right] = \epsilon$.

Here the probability is measured over the random choices of $a, b, c \in \mathbb{Z}_q^*$ and the random bits of $\mathcal{A}$.

Generalized Bilinear Diffie-Hellman Problem (GBDHP): Let $\mathbb{G}_1$, $\mathbb{G}_2$, P and e be as above. The GBDHP in $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$ is as follows: Given $\langle P, aP, bP, cP \rangle$ with uniformly random choices of $a, b, c \in \mathbb{Z}_q^*$, output a pair $\langle Q \in \mathbb{G}_1^*, e(P, Q)^{abc} \in \mathbb{G}_2 \rangle$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the GBDHP in $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$ if
$\Pr\left[\mathcal{A}(\langle P, aP, bP, cP \rangle) = \langle Q, e(P, Q)^{abc} \rangle\right] = \epsilon$.

Here the probability is measured over the random choices of $a, b, c \in \mathbb{Z}_q^*$ and the random bits of $\mathcal{A}$.

Notice that the BDHP is a special case of the GBDHP in which the algorithm outputs the choice $Q = P$. While the GBDHP may appear to be in general easier to solve than the BDHP because the solver gets to choose Q, we know of no polynomial-time algorithm for solving either when the groups $\mathbb{G}_1, \mathbb{G}_2$ and pairing e are appropriately selected. If the solver knows $s \in \mathbb{Z}_q^*$ such that $Q = sP$, then the problems are of course equivalent.

BDH Parameter Generator: As in [5], the formal output of this randomized algorithm is a triple $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are of prime order q and $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a pairing.

Our security proofs will yield reductions to the BDHP or GBDHP in groups generated by a BDH parameter generator $\mathcal{IG}$.

3 Certificateless Public Key Encryption

In this section we present a formal definition for a certificateless public key encryption (CL-PKE) scheme. We also examine the capabilities which may be possessed by the adversaries against such a scheme and give a security model for CL-PKE.

A CL-PKE scheme is specified by seven randomized algorithms.

`Setup`: This algorithm takes security parameter k and returns the system parameters `params` and `master-key`. The system parameters includes a description of the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Usually, this algorithm is run by the KGC. We assume throughout that `params` are publicly and authentically available, but that only the KGC knows `master-key`.

`Partial-Private-Key-Extract`: This algorithm takes `params`, `master-key` and an identifier for entity A, $\mathtt{ID}_A \in \{0,1\}^*$, as input. It returns a partial private key D_A. Usually this algorithm is run by the KGC and its output is transported to entity A over a confidential and authentic channel.

`Set-Secret-Value`: This algorithm takes as inputs `params` and an entity A's identifier $\mathtt{ID}_A$ as inputs and outputs A's secret value x_A.

`Set-Private-Key`: This algorithm takes `params`, an entity A's partial private key D_A and A's secret value x_A as input. The value x_A is used to transform D_A into the (full) private key S_A. The algorithm returns S_A.

`Set-Public-Key`: This algorithm takes `params` and entity A's secret value x_A as input and from these constructs the public key P_A for entity A.

Normally both `Set-Private-Key` and `Set-Public-Key` are run by an entity A for itself, after running `Set-Secret-Value`. The same secret value x_A is used in each. Separating them makes it clear that there is no need for a temporal ordering on the generation of public and private keys in our CL-PKE scheme. Usually, A is the only entity in possession of S_A and x_A, and x_A will be chosen at random from a suitable and large set.

`Encrypt`: This algorithm takes as inputs `params`, a message $M \in \mathcal{M}$, and the public key P_A and identifier $\mathtt{ID}_A$ of an entity A. It returns either a ciphertext $C \in \mathcal{C}$ or the null symbol $\perp$ indicating an encryption failure. This will always occur in the event that P_A does not have the correct form. In our scheme, this is the only way an encryption failure will occur.

`Decrypt`: This algorithm takes as inputs `params`, $C \in \mathcal{C}$, and a private key S_A. It returns a message $M \in \mathcal{M}$ or a message $\perp$ indicating a decryption failure.

Naturally, we insist that output M should result from applying algorithm `Decrypt` with inputs `params`, S_A on a ciphertext C generated by using algorithm `Encrypt` with inputs `params`, P_A, $\mathtt{ID}_A$ on message M.

3.1 Security Model for CL-PKE

Given this formal definition of a CL-PKE scheme, we are now in a position to define adversaries for such a scheme. The standard definition for security for a public key encryption scheme involves indistinguishability of encryptions against a fully-adaptive chosen ciphertext attacker (IND-CCA) [4, 10, 20]. In this definition, there are two parties, the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$. The adversary operates in three phases after being presented with a random public key. In Phase 1, $\mathcal{A}$ may make decryption queries on ciphertexts of its choice. In the Challenge Phase, $\mathcal{A}$ chooses two messages M_0, M_1 and is given a challenge ciphertext C^* for one of these two messages M_b by the challenger. In Phase 2, $\mathcal{A}$ may make further decryption queries, but may not ask for the decryption of C^*. The attack ends with $\mathcal{A}$'s guess b' for the bit b. The adversary's advantage is defined to be $\mathrm{Adv}(\mathcal{A}) = 2(\Pr[b' = b] - \frac{1}{2})$.

This model was strengthened for ID-PKC in [5] to handle adversaries who can extract the private keys of arbitrary entities and who choose the identity $\mathtt{ID}_{\mathrm{ch}}$ of the entity on whose public key they are challenged. This extension is appropriate because the compromise of some entities' private keys should not affect the security of an uncompromised entity's encryptions.

Here, we extend the model of [5] to allow adversaries who can extract partial private keys, or private keys, or both, for identities of their choice. Given that our scheme has no certificates, we must further strengthen the model to allow for adversaries who can replace the public key of any entity with a value of their choice. We must also consider carefully how a challenger should respond to key extraction and decryption queries for identities whose public keys have been changed.

Here then is a list of the actions that a general adversary against a CL-PKE scheme may carry out and a discussion of each action.
(1) **Extract partial private key of A:** $\mathcal{C}$ responds by running algorithm `Partial-Private-Key-Extract` to generate the partial private key D_A for entity A.
(2) **Extract private key for A:** As in [5], we allow our adversary $\mathcal{A}$ to make requests for entities' private keys. If A's public key has not been replaced then $\mathcal{C}$ can respond by running algorithm `Set-Private-Key` to generate the private key S_A for entity A (first running `Set-Secret-Value` for A if necessary). But it is unreasonable to expect $\mathcal{C}$ to be able to respond to such a query if $\mathcal{A}$ has already replaced A's public key. Of course, we insist that $\mathcal{A}$ does not at any point extract the private key for the selected challenge identity $\mathtt{ID}_{\mathrm{ch}}$.
(3) **Request public key of A:** Naturally, we assume that public keys are available to $\mathcal{A}$. On receiving a first request for A's public key, $\mathcal{C}$ responds by running algorithm `Set-Public-Key` to generate the public key P_A for entity A (first running `Set-Secret-Value` for A if necessary).
(4) **Replace public key of A:** $\mathcal{A}$ can repeatedly replace the public key P_A for any entity A with any value P'_A of its choice. In our concrete CL-PKE schemes, our public keys will have a certain structure that is used to test the validity of public keys before any encryption. We assume here that the adversary's choice P'_A is a valid public key; this assumption can be removed (and our schemes remain secure) at the cost of some additional complexity in our definitions. Note that in our schemes, any entity can easily create public keys that are valid. The current value of an entity's public key is used by $\mathcal{C}$ in any computations (for example, preparing a challenge ciphertext) or responses to $\mathcal{A}$'s requests (for example, replying to a request for the public key). We insist that $\mathcal{A}$ cannot both replace the public key for the challenge identity $\mathtt{ID}_{\mathrm{ch}}$ before the challenge phase *and* extract the partial private key for $\mathtt{ID}_{\mathrm{ch}}$ in some phase – this would enable $\mathcal{A}$ to receive a challenge ciphertext under a public key for which it could compute the private key.
(5) **Decryption query for ciphertext C and entity A:** If $\mathcal{A}$ has not replaced the public key of entity A, then $\mathcal{C}$ responds by running the algorithm `Set-Private-Key` to obtain the private key S_A, then running `Decrypt` on ciphertext C and private key S_A and returning the output to $\mathcal{A}$. However, if $\mathcal{A}$

has already replaced the public key of A, then in following this approach, $\mathcal{C}$ would (in general) not decrypt using a private key matching the current public key. However, we insist that $\mathcal{C}$ properly decrypts ciphertexts even for entities whose public keys have been replaced (these decryptions will be handled using special purpose knowledge extractors in our security proofs). This results in a very powerful security model because decryption queries made under public keys that have been changed are potentially far more useful to $\mathcal{A}$. Naturally, as in [5], we prohibit $\mathcal{A}$ from ever making a decryption query on the challenge ciphertext C^* for the combination of identity $\mathtt{ID}_{\mathrm{ch}}$ and public key P_{ch} that was used to encrypt M_b. However $\mathcal{A}$ is, for example, allowed to replace the public key for $\mathtt{ID}_{\mathrm{ch}}$ with a new value and then request a decryption of C^*, or to change another entity A's public key to P_{ch} (or any other value) and then request the decryption of C^* for entity A.

We also want to consider adversaries who are equipped with `master-key`, in order to model security against an eavesdropping KGC. As discussed in Section 1, we do not allow such an adversary to replace public keys: in this respect, we invest in the KGC the same level of trust as we do in a CA in a traditional PKI. So we will distinguish between two adversary types, with slightly different capabilities:

CL-PKE Type I Adversary: Such an adversary $\mathcal{A}_I$ does not have access to `master-key`. However, $\mathcal{A}_I$ may request public keys and replace public keys with values of its choice, extract partial private and private keys and make decryption queries, all for identities of its choice. As discussed above, we make several natural restrictions on such a Type I adversary: (1) $\mathcal{A}_I$ cannot extract the private key for $\mathtt{ID}_{\mathrm{ch}}$ at any point. (2) $\mathcal{A}_I$ cannot request the private key for any identity if the corresponding public key has already been replaced. (3) $\mathcal{A}_I$ cannot both replace the public key for the challenge identity $\mathtt{ID}_{\mathrm{ch}}$ before the challenge phase *and* extract the partial private key for $\mathtt{ID}_{\mathrm{ch}}$ in some phase. (4) In Phase 2, $\mathcal{A}_I$ cannot make a decryption query on the challenge ciphertext C^* for the combination of identity $\mathtt{ID}_{\mathrm{ch}}$ and public key P_{ch} that was used to encrypt M_b.

CL-PKE Type II Adversary: Such an adversary $\mathcal{A}_{II}$ does have access to `master-key`, but may not replace public keys of entities. Adversary $\mathcal{A}_{II}$ can compute partial private keys for itself, given `master-key`. It can also request public keys, make private key extraction queries and decryption queries, both for identities of its choice. The restrictions on this type of adversary are: (1) $\mathcal{A}_{II}$ cannot replace public keys at any point. (2) $\mathcal{A}_{II}$ cannot extract the private key for $\mathtt{ID}_{\mathrm{ch}}$ at any point. (3) In Phase 2, $\mathcal{A}_{II}$ cannot make a decryption query on the challenge ciphertext C^* for the combination of identity $\mathtt{ID}_{\mathrm{ch}}$ and public key P_{ch} that was used to encrypt M_b.

Chosen ciphertext security for CL-PKE: We say that a CL-PKE scheme is semantically secure against an adaptive chosen ciphertext attack ("IND-CCA secure") if no polynomially bounded adversary $\mathcal{A}$ of Type I or Type II has a non-negligible advantage against the challenger in the following game:

Setup: The challenger takes a security parameter k and runs the `Setup` algorithm. It gives $\mathcal{A}$ the resulting system parameters `params`. If $\mathcal{A}$ is of Type I, then the challenger keeps `master-key` to itself, otherwise, it gives `master-key` to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ issues a sequence of requests, each request being either a partial private key extraction, a private key extraction, a request for a public key, a replace public key command or a decryption query for a particular entity. These queries may be asked adaptively, but are subject to the rules on adversary behaviour defined above.

Challenge Phase: Once $\mathcal{A}$ decides that Phase 1 is over it outputs the challenge identity $\mathtt{ID}_{\mathrm{ch}}$ and two equal length plaintexts $M_0, M_1 \in \mathcal{M}$. Again, the adversarial constraints given above apply. The challenger now picks a random bit $b \in \{0,1\}$ and computes C^*, the encryption of M_b under the current public key P_{ch} for $\mathtt{ID}_{\mathrm{ch}}$. If the output of the encryption is $\perp$, then $\mathcal{A}$ has immediately lost the game (it has replaced a public key with one not having the correct form). Otherwise, C^* is delivered to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ issues a second sequence of requests as in Phase 1, again subject to the rules on adversary behaviour above.

Guess: Finally, $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The adversary wins the game if $b = b'$. We define $\mathcal{A}$'s advantage in this game to be $\mathrm{Adv}(\mathcal{A}) := 2(\Pr[b = b'] - \frac{1}{2})$.

4 CL-PKE Schemes from Pairings

In this section, we describe a pair of CL-PKE schemes. Our first scheme, `BasicCL-PKE`, is analogous to the scheme `BasicIdent` of [5], and is included only to serve as a warm-up for our main scheme `FullCL-PKE`. The main scheme is in turn an analogue of the scheme `FullIdent` of [5] and is IND-CCA secure, assuming the hardness of the GBDHP. We prove this in Theorem 1.

4.1 A Basic CL-PKE Scheme

We describe the seven algorithms needed to define `BasicCL-PKE`. We let k be a security parameter given to the `Setup` algorithm and $\mathcal{IG}$ a BDH parameter generator with input k.

`Setup`: This algorithm runs as follows:
(1) Run $\mathcal{IG}$ on input k to generate output $\langle \mathbb{G}_1, \mathbb{G}_2, e\rangle$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of some prime order q and $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a pairing. (2) Choose an arbitrary generator $P \in \mathbb{G}_1$. (3) Select a `master-key` s uniformly at random from $\mathbb{Z}_q^*$ and set $P_0 = sP$. (4) Choose cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^n$. Here n will be the bit-length of plaintexts.

The system parameters are `params`$= \langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2\rangle$. The `master-key` is $s \in \mathbb{Z}_q^*$. The message space is $\mathcal{M} = \{0,1\}^n$ and the ciphertext space is $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^n$.

`Partial-Private-Key-Extract`: This algorithm takes as input an identifier

$\mathtt{ID}_A \in \{0,1\}^*$, and carries out the following steps to construct the partial private key for entity A with identifier $\mathtt{ID}_A$:
(1) Compute $Q_A = H_1(\mathtt{ID}_A) \in \mathbb{G}_1^*$.
(2) Output the partial private key $D_A = sQ_A \in \mathbb{G}_1^*$.

The reader will notice that the partial private key of entity A here is identical to that entity's private key in the schemes of [5]. Also notice that A can verify the correctness of the `Partial-Private-Key-Extract` algorithm output by checking $e(D_A, P) = e(Q_A, P_0)$.

`Set-Secret-Value`: This algorithm takes as inputs `params` and an entity A's identifier $\mathtt{ID}_A$ as inputs. It selects $x_A \in \mathbb{Z}_q^*$ at random and outputs x_A as A's secret value.

`Set-Private-Key`: This algorithm takes as inputs `params`, an entity A's partial private key D_A and A's secret value $x_A \in \mathbb{Z}_q^*$. It transforms partial private key D_A to private key S_A by computing $S_A = x_A D_A = x_A s Q_A \in \mathbb{G}_1^*$.

`Set-Public-Key`: This algorithm takes `params` and entity A's secret value $x_A \in \mathbb{Z}_q^*$ as inputs and constructs A's public key as $P_A = \langle X_A, Y_A \rangle$, where $X_A = x_A P$ and $Y_A = x_A P_0 = x_A s P$.

`Encrypt`: To encrypt $M \in \mathcal{M}$ for entity A with identifier $\mathtt{ID}_A \in \{0,1\}^*$ and public key $P_A = \langle X_A, Y_A \rangle$, perform the following steps:
(1) Check that $X_A, Y_A \in \mathbb{G}_1^*$ and that the equality $e(X_A, P_0) = e(Y_A, P)$ holds. If not, output $\perp$ and abort encryption. (2) Compute $Q_A = H_1(\mathtt{ID}_A) \in \mathbb{G}_1^*$. (3) Choose a random value $r \in \mathbb{Z}_q^*$. (4) Compute and output the ciphertext: $C = \langle rP, M \oplus H_2(e(Q_A, Y_A)^r) \rangle$.

Notice that this encryption operation is identical to the encryption algorithm in the scheme `BasicIdent` of [5], except for the check on the structure of the public key in step 1 and the use of Y_A in place of $P_0 = P_{pub}$ in step 4.

`Decrypt`: Suppose $C = \langle U, V \rangle \in \mathcal{C}$. To decrypt this ciphertext using the private key S_A, compute and output: $V \oplus H_2(e(S_A, U))$.

Notice that if $\langle U = rP, V \rangle$ is the encryption of M for entity A with public key $P_A = \langle X_A, Y_A \rangle$, the decryption is the inverse of encryption.

Again, the similarity to the decryption operation of `BasicIdent` should be apparent.

We have presented this scheme to help the reader understand our `FullCL-PKE` scheme, and so we do not analyse its security in detail.

4.2 A Full CL-PKE Scheme

Now that we have described our basic CL-PKE scheme, we add chosen ciphertext security to it, adapting the Fujisaki-Okamoto padding technique [11]. The algorithms for `FullCL-PKE` are as follows:

`Setup`: Identical to `Setup` for `BasicCL-PKE`, except that we choose two additional cryptographic hash functions $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and $H_4 : \{0,1\}^n \to \{0,1\}^n$.

The system parameters are `params`= $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2, H_3, H_4 \rangle$. The `master-key` and message space $\mathcal{M}$ are the same as in `BasicCL-PKE`. The ciphertext space is now $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^{2n}$.

`Partial-Private-Key-Extract`, `Set-Secret-Value`, `Set-Private-Key`, and `Set-Public-Key`: Identical to `BasicCL-PKE`.

`Encrypt`: To encrypt $M \in \mathcal{M}$ for entity A with identifier $\mathtt{ID}_A \in \{0,1\}^*$ and public key $P_A = \langle X_A, Y_A \rangle$, perform the following steps:
(1) Check that $X_A, Y_A \in \mathbb{G}_1^*$ and that the equality $e(X_A, P_0) = e(Y_A, P)$ holds. If not, output $\perp$ and abort encryption.
(2) Compute $Q_A = H_1(\mathtt{ID}) \in \mathbb{G}_1^*$.
(3) Choose a random $\sigma \in \{0,1\}^n$.
(4) Set $r = H_3(\sigma, M)$.
(5) Compute and output: $C = \langle rP, \sigma \oplus H_2(e(Q_A, Y_A)^r), M \oplus H_4(\sigma) \rangle$.

`Decrypt`: Suppose the ciphertext $C = \langle U, V, W \rangle \in \mathcal{C}$. To decrypt this ciphertext using the private key S_A:
(1) Compute $V \oplus H_2(e(S_A, U)) = \sigma'$.
(2) Compute $W \oplus H_4(\sigma') = M'$.
(3) Set $r' = H_3(\sigma', M')$ and test if $U = r'P$. If not, output $\perp$ and reject C.
(4) Output M' as the decryption of C.

When C is a valid encryption of M using P_A and $\mathtt{ID}_A$, it is easy to see that decrypting C will result in an output $M' = M$. We note that W can be replaced by $W = E_{H_4(\sigma)}(M)$ where E denotes a semantically secure symmetric key encryption scheme as in [11] (though our security proofs will require some modifications to handle this case). This concludes the description of `FullCL-PKE`.

4.3 Security of the Scheme `FullCL-PKE`

We have the following theorem about the security of `FullCL-PKE`.

Theorem 1 *Let hash functions H_1, H_2, H_3 and H_4 be random oracles. Suppose further that there is no polynomially bounded algorithm that can solve the GBDHP in groups generated by $\mathcal{IG}$ with non-negligible advantage. Then* `FullCL-PKE` *is IND-CCA secure.*

This theorem follows from a sequence of lemmas that are proved in the appendices. It can be made into a concrete security reduction relating the advantage ϵ of a Type I or Type II attacker against `FullCL-PKE` to that of an algorithm to solve GBDHP or BDHP.

5 Further CL-PKC Schemes

In this section, we sketch another CL-PKC primitives: a signature scheme based on the identity-based scheme of [17]. We begin by outlining an alternative key generation technique which enhances the resilience of our schemes against a cheating KGC and allows for non-repudation of certificateless signatures.

5.1 An Alternative Key Generation Technique

Up to this point, we have assumed that the KGC is trusted to not replace the public keys of users and to only issue one copy of each partial private key, to the correct recipient. This may involve an unacceptable level of trust in the KGC for some users. Our current set up also allows users to create more than one public key for the same partial private key. This can be desirable in some applications, but undesirable in others.

Here we sketch a simple binding technique which ensures that users can only create one public key for which they know the corresponding private key. In our technique, an entity A must first fix its secret value x_A and its public key $P_A = \langle X_A, Y_A \rangle$. We then re-define Q_A to be $Q_A = H_1(\mathtt{ID}_A \| P_A)$ – now Q_A binds A's identifier and public key. The partial private key delivered to entity A is still $D_A = sQ_A$ and the private key created by A is still xsQ_A. However, these are also now bound to A's choice of public key. This binding effectively restricts A to using a single public key, since A can now only compute one private key from D_A.

This technique has a very important additional benefit: it reduces the degree of trust that users need to have in the KGC in our certificateless schemes. In short, the technique raises our schemes to trust level 3 in the trust hierarchy of [15], the same level as is enjoyed in a traditional PKI. Now, with our binding technique in place, a KGC who replaces an entity's public key will be implicated in the event of a dispute: the existence of two working public keys for an identity can only result from the existence of two partial private keys binding that identity to two different public keys; only the KGC could have created these two partial private keys. Thus our binding technique makes the KGC's replacement of a public key apparent and equivalent to a CA forging a certificate in a traditional PKI.

Theorem 1 still applies for our CL-PKE scheme with this binding in place because of the way in which H_1 is modelled as a random oracle. Notice too that with this binding in place, there is no longer any need to keep partial private keys secret: informally, knowledge of the key $D_A = sQ_A$ does not help an adversary to create the unique private key $S_A = xsQ_A$ that matches the particular public key P_A that is bound to D_A. When applied to the certificateless signature primitive in this section, the binding technique ensures a stronger form of non-repudiation: without the binding, an entity could always attempt to repudiate a signature by producing a second working public key and claiming that the KGC had created the signature using the first public key.

Even with this binding in place, the security analysis of our original encryption scheme (in which an adversary can replace public keys) is still important: it models the scenario where an adversary *temporarily* replaces the public key P_A of an entity A with a new value P'_A in an attempt to obtain a ciphertext which he can distinguish, and then resets the public key. In this case, our proof shows that the adversary does not gain any advantage in a distinguishing game unless he has access to the matching partial private key $D'_A = sH_1(\mathtt{ID}_A \| P'_A)$. In turn, this partial private key should not be made available by the KGC. Of course, nothing can prevent a KGC from mounting an attack of this type, but the same applies for the CA in a traditional PKI.

5.2 A Certificateless Signature Scheme

We will describe a certificateless public-key signature (CL-PKS) scheme that is based on a provably secure ID-PKC signature scheme of [17].

In general, a CL-PKS scheme can be specified by seven algorithms: `Setup`, `Partial-Private-Key-Extract`, `Set-Secret-Value`, `Set-Private-Key`, `Set-Public-Key`, `Sign` and `Verify`. These are similar to the algorithms used to define a CL-PKE scheme: `Setup` and `params` are modified to include a description of the signature space $\mathcal{S}$, `Partial-Private-Key-Extract`, `Set-Secret-Value`, `Set-Private-Key` and `Set-Public-Key` are just as before and `Sign` and `Verify` are as follows:

`Sign`: This algorithm takes as inputs `params`, a message $M \in \mathcal{M}$ to be signed and a private key S_A. It outputs a signature $Sig \in \mathcal{S}$.

`Verify`: This algorithm takes as inputs `params`, a message $M \in \mathcal{M}$, the identifier $\mathtt{ID}_A$ and public key P_A of an entity A, and $Sig \in \mathcal{S}$ as the signature to be verified. It outputs `valid`, `invalid` or $\perp$.

Given this general description, we now outline a CL-PKS scheme:

`Setup`: This is identical to `Setup` for our scheme `BasicCL-PKE`, except that now there is only one hash function $H : \{0,1\}^* \times \mathbb{G}_2 \to \mathbb{Z}_q^*$ and `params` is $\langle \mathbb{G}_1, \mathbb{G}_2, n, e, P, P_0, H \rangle$. The signature space is defined as $\mathcal{S} = \mathbb{G}_1 \times \mathbb{Z}_q^*$.

`Partial-Private-Key-Extract`, `Set-Secret-Value`, `Set-Private-Key` and `Set-Public-Key`: Identical to `BasicCL-PKE`.

`Sign`: To sign $M \in \mathcal{M}$ using the private key S_A, perform the following steps: (1) Choose random $a \in \mathbb{Z}_q^*$. (2) Compute $r = e(P,P)^a \in \mathbb{G}_2$. (3) Set $v = H(M,r) \in \mathbb{Z}_q^*$. (4) Compute $U = vS_A + aP \in \mathbb{G}_1$. (5) Output as the signature $\langle U, v \rangle$.

`Verify`: To verify a purported signature $\langle U, v \rangle$ on a message $M \in \mathcal{M}$ for identity $\mathtt{ID}_A$ and public key $\langle X_A, Y_A \rangle$: (1) Check that the equality $e(X_A, P_0) = e(Y_A, P)$ holds. If not, output $\perp$ and abort verification. (2) Compute $r = e(U,P) \cdot e(Q_A, -Y_A)^v$. (3) Check if $v = H(M,r)$ holds. If it does, output `valid`, otherwise output `invalid`.

5.3 Other Schemes

The hierarchical encryption and signature schemes of [14] and the key agreement scheme of [23] can be adapted to our certificateless setting. These adaptations are presented in the full paper [2].

6 Conclusions

In this paper we introduced the concept of *certificateless public key cryptography*, a model for the use of public key cryptography that is intermediate between the identity-based approach and traditional PKI. We showed how our concept can be realized by specifying a certificateless public key encryption (CL-PKE) scheme that is based on bilinear maps. We showed that our CL-PKE scheme is secure in an appropriate model, assuming that the Generalized Bilinear Diffie-Hellman Problem (GBDHP) is hard. We also rounded out our treatment by briefly presenting a certificateless signature scheme.

Acknowledgement

We would like to thank Dan Boneh, Alex Dent, Steven Galbraith and Craig Gentry for their comments and helpful discussions on the paper.

References

[1] C. Adams and S. Lloyd. *Understanding Public-Key Infrastructure – Concepts, Standards, and Deployment Considerations*. Macmillan, Indianapolis, USA, 1999.

[2] S.S. Al-Riyami and K.G. Paterson. Certificateless public key cryptography. Cryptology ePrint Archive, Report 2003/126, 2003. `http://eprint.iacr.org/`.

[3] P.S.L.M. Barreto *et al.* Efficient algorithms for pairing-based cryptosystems. In *Proc. CRYPTO 2002*, LNCS vol. 2442, pp. 354–368. Springer, 2002.

[4] M. Bellare *et al.* Relations among notions of security for public-key encryption schemes. In *Proc. CRYPTO 1998*, LNCS vol. 1462. Springer, 1998.

[5] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Proc. CRYPTO 2001*, LNCS vol. 2139, pp. 213–229. Springer, 2001.

[6] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Computing*, 32(3):586–615, 2003.

[7] D. Boneh, H. Shacham, and B. Lynn. Short signatures from the Weil pairing. In C. Boyd, editor, *Proc. ASIACRYPT 2001*, LNCS vol. 2248, pp. 514–532. Springer, 2001.

[8] L. Chen *et al.* Certification of public keys within an identity based system. In A. H. Chan and V. D. Gligor, editors, *Information Security, 5th International Conference, ISC*, LNCS vol. 2433, pp. 322–333. Springer, 2002.

[9] J. Dankers *et al.* Public key infrastructure in mobile systems. *IEE Electronics and Commucation Engineering Journal*, 14(5):180–190, 2002.

[10] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal of Computing*, 30(2):391–437, 2000.

[11] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. J. Wiener, editor, *Proc. CRYPTO 1999*, LNCS vol. 1666, pp. 537–554. Springer, 1999.

[12] S.D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithmic Number Theory 5th International Symposium, ANTS-V*, LNCS vol. 2369, pp. 324–337. Springer, 2002.

[13] C. Gentry. Certificate-based encryption and the certificate revocation problem. In E. Biham, editor, *Proc. EUROCRYPT 2003*, LNCS vol. 2656, pp. 272–293. Springer, 2003.
[14] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In Y. Zheng, editor, *Proc. ASIACRYPT 2002*, LNCS vol. 2501, pp. 548–566. Springer, 2002.
[15] M. Girault. Self-certified public keys. In D. W. Davies, editor, *Proc. EUROCRYPT 1991*, LNCS vol. 547, pp. 490–497. Springer, 1992.
[16] P. Gutmann. PKI: It's not dead, just resting. *IEEE Computer*, 35(8):41–49, 2002.
[17] F. Hess. Efficient identity based signature schemes based on pairings. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography 9th Annual International Workshop, SAC 2002*, LNCS vol. 2595, pp. 310–324. Springer, 2003.
[18] K.G. Paterson. Cryptography from pairings: a snapshot of current research. *Information Security Technical Report*, 7(3):41–54, 2002.
[19] H. Petersen and P. Horster. Self-certified keys – concepts and applications. In *3rd Int. Conference on Communications and Multimedia Security*. Chapman and Hall, 1997.
[20] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attacks. In *Proc. CRYPTO 1991*, LNCS vol. 576, pp. 433–444. Springer, 1991.
[21] S. Saeednia. Identity-based and self-certified key-exchange protocols. In V. Varadharajan, J. Pieprzyk, and Y. Mu, editors, *Information Security and Privacy, Second Australasian Conference, ACISP*, LNCS vol. 1270, pp. 303–313. Springer, 1997.
[22] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proc. CRYPTO 1984*, LNCS vol. 196, pp. 47–53. Springer, 1984.
[23] N.P. Smart. An identity based authenticated key agreement protocol based on the Weil pairing. *Electronics Letters*, 38(13):630–632, 2002.
[24] N.P. Smart. Access control using pairing based cryptography. In M. Joye, editor, *Proceedings CT-RSA 2003*, LNCS vol. 2612, pp. 111–121. Springer, 2003.

Appendix A: Proofs of Security for `FullCL-PKE`

A.1 Two Public Key Encryption Schemes

We define a public key encryption scheme `HybridPub`. It will be used as a tool in our security proof for `FullCL-PKE`.

`HybridPub`**:** This scheme is specified by three algorithms: `Key-Generation`, `Encrypt` and `Decrypt`.

`Key-Generation`**:** (1) Run $\mathcal{IG}$ to generate $\langle \mathbb{G}_1, \mathbb{G}_2, e\rangle$ with the usual properties. Choose a generator $P \in \mathbb{G}_1$. (2) Pick a random $Q \in \mathbb{G}_1^*$, a random $s \in \mathbb{Z}_q^*$ and a random $x \in \mathbb{Z}_q^*$. (3) Set $P_0 = sP$, $X = xP$, $Y = xsP$ and $S = xsQ$. (4) Choose the cryptographic hash functions $H_2 : \mathbb{G}_2 \to \{0,1\}^n$, $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and $H_4 : \{0,1\}^n \to \{0,1\}^n$. The public key is $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, X, Y, Q, H_2, H_3, H_4\rangle$. The private key is $S = xsQ$, the message space is $\mathcal{M} = \{0,1\}^n$ and the ciphertext space is $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^{2n}$.

`Encrypt`**:** To encrypt $M \in \mathcal{M}$, perform the following steps: (1) Check that the equality $e(X, P_0) = e(Y, P)$ holds. If not, output $\perp$ and abort encryption. (2) Choose a random $\sigma \in \{0,1\}^n$. (3) Set $r = H_3(\sigma, M)$. (4) Compute and output the ciphertext: $C = \langle rP, \sigma \oplus H_2(e(Q, Y)^r), M \oplus H_4(\sigma)\rangle$.

`Decrypt`**:** To decrypt $C = \langle U, V, W\rangle \in \mathcal{C}$ using private key S, do the following: (1) Compute $V \oplus H_2(e(S, U)) = \sigma'$. (2) Compute $W \oplus H_4(\sigma') = M'$. (3) Set $r' = H_3(\sigma', M')$ and test if $U = r'P$. If not, output $\perp$ and reject the ciphertext. (4) Output M' as the decryption of C.

A second scheme `BasicPub` is defined in [2]; it is a simplified version of `HybridPub` in which the encryption of message M equals $\langle rP, M \oplus H_2(e(Q,Y)^r)\rangle$. The full paper [2] also defines Type I and II IND-CCA, IND-CPA and OWE adversaries for `BasicPub` and `HybridPub`: these are similar to the usual definitions, except that a Type I adversary is allowed to replace the public key, while a Type II adversary has the value s.

A.2: Statements of Lemmas

We present a series of lemmas. Theorem 1 for Type I adversaries follows by combining Lemmas 2, 3, 4, 5 and 8. Similarly, Theorem 1 for Type II adversaries follows by combining Lemmas 6, 7 and 8.

Lemma 2 *Suppose that H_1, H_2, H_3 and H_4 are random oracles and that there exists an IND-CCA Type I adversary $\mathcal{A}_I$ against* `FullCL-PKE`. *Suppose $\mathcal{A}_I$ has advantage ϵ, runs in time t, makes q_i queries to H_i ($1 \le i \le 4$) and makes q_d decryption queries. Then there is an algorithm $\mathcal{B}$ which acts as either a Type I or a Type II IND-CPA adversary against* `HybridPub`. *Moreover, $\mathcal{B}$ either has advantage at least $\epsilon\lambda^{q_d}/4q_1$ when playing as a Type I adversary, or has advantage at least $\epsilon\lambda^{q_d}/4q_1$ when playing as a Type II adversary. $\mathcal{B}$ runs in time $t + O((q_3+q_4)q_d t')$. Here t' is the running time of the* `BasicCL-PKE` *encryption algorithm and*

$$\begin{aligned} 1-\lambda \quad &\le \quad (q_3+q_4)\cdot\epsilon_{OWE}(t+O((q_3+q_4)q_d t', q_2) \\ &\qquad +\epsilon_{GBDHP}(t+O((q_3+q_4)q_d t') + 3q^{-1} + 2^{-n+1}, \end{aligned}$$

where $\epsilon_{OWE}(T,q')$ denotes the highest advantage of any Type I or Type II OWE adversary against `BasicPub` *which operates in time T and makes q' hash queries to H_2, and $\epsilon_{GBDHP}(T)$ denotes the highest advantage of any time T algorithm to solve GBDHP in groups of order q generated by $\mathcal{IG}$.*

Lemma 3 *Suppose that H_3 and H_4 are random oracles. Let $\mathcal{A}_I$ be a Type I IND-CPA adversary against* `HybridPub` *which has advantage ϵ and makes q_4 queries to H_4. Then there exists a Type I OWE adversary $\mathcal{A}'_I$ against* `BasicPub` *which runs in time $O(time(\mathcal{A}_I))$ and has advantage at least $\epsilon/2(q_3+q_4)$.*

Lemma 4 *Suppose that H_3 and H_4 are random oracles. Let $\mathcal{A}_I$ be a Type II IND-CPA adversary against* `HybridPub` *which has advantage ϵ and makes q_4 queries to H_4. Then there exists a Type II OWE adversary $\mathcal{A}'_I$ against* `BasicPub` *which runs in time $O(time(\mathcal{A}_{II}))$ and has advantage at least $\epsilon/2(q_3+q_4)$.*

Lemma 5 *Suppose that H_2 is a random oracle. Suppose there exists a Type I OWE adversary $\mathcal{A}_I$ against* `BasicPub` *which makes at most q_2 queries to H_2 and which has advantage ϵ. Then there exists an algorithm $\mathcal{B}$ to solve the GBDHP which runs in time $O(time(\mathcal{A}_I))$ and has advantage at least $(\epsilon - \frac{1}{2^n})/q_2$.*

Lemma 6 *Suppose that H_1 is a random oracle and that there exists an IND-CCA Type II adversary $\mathcal{A}_{II}$ on* `FullCL-PKE` *with advantage ϵ which makes at most q_1 queries to H_1. Then there is an IND-CCA Type II adversary on* `HybridPub` *with advantage at least ϵ/q_1 which runs in time $O(time(\mathcal{A}_{II}))$.*

Lemma 7 *Suppose that H_3 and H_4 are random oracles. Let $\mathcal{A}_{II}$ be a Type II IND-CCA adversary against* `HybridPub` *which has advantage ϵ, makes q_d decryption queries, q_3 queries to H_3 and q_4 queries to H_4. Then there exists a Type II OWE adversary $\mathcal{A}'_{II}$ against* `BasicPub` *with*

$$\begin{array}{rcl} time(\mathcal{A}'_{II}) & = & time(\mathcal{A}_{II}) + O(n(q_3+q_4)) \\ Adv(\mathcal{A}'_{II}) & \geq & \frac{1}{2(q_3+q_4)}\left((\epsilon+1)(1-q^{-1}-2^{-n})^{q_d}-1\right). \end{array}$$

Lemma 8 *Suppose that H_2 is a random oracle. Suppose there exists a Type II OWE adversary $\mathcal{A}_{II}$ against* `BasicPub` *which makes at most q_2 queries to H_2 and which has advantage ϵ. Then there exists an algorithm $\mathcal{B}$ to solve the BDHP which runs in time $O(time(\mathcal{A}_{II}))$ and has advantage at least $(\epsilon - \frac{1}{2^n})/q_2$.*

A.3: Proofs of Lemmas

Proof of Lemma 2: Let $\mathcal{A}_I$ be a Type I IND-CCA adversary against `FullCL-PKE`. Suppose $\mathcal{A}_I$ has advantage ϵ, runs in time t, makes q_i queries to random oracle H_i ($1 \leq i \leq 4$) and makes q_d decryption queries. We show how to construct from $\mathcal{A}_I$ an adversary $\mathcal{B}$ that acts either as a Type I IND-CCA adversary against `HybridPub` or as a Type II IND-CCA adversary against `HybridPub`. We assume that challengers $\mathcal{C}_I, \mathcal{C}_{II}$ for both types of game are available to $\mathcal{B}$.

Adversary $\mathcal{B}$ begins by choosing a random bit c and an index I uniformly at random with $1 \leq I \leq q_1$. If $c = 0$, then $\mathcal{B}$ chooses to play against $\mathcal{C}_I$ and aborts $\mathcal{C}_{II}$. Here, $\mathcal{B}$ will build a Type I IND-CPA adversary against `HybridPub` and fails against $\mathcal{C}_{II}$. When $c = 1$, $\mathcal{B}$ chooses to play against $\mathcal{C}_{II}$ and aborts $\mathcal{C}_I$. Here, $\mathcal{B}$ will build a Type II IND-CPA adversary against `HybridPub` and fails against $\mathcal{C}_I$. In either case, $\mathcal{C}$ will denote the challenger against which $\mathcal{B}$ plays for the remainder of this proof. We let $\mathcal{H}$ denote the event that $\mathcal{A}_I$ chooses $\mathtt{ID}_I$ as the challenge identity $\mathtt{ID}_{\mathrm{ch}}$. We let $\mathcal{F}_0$ denote the event that $\mathcal{A}_I$ extracts the partial private key for entity $\mathtt{ID}_I$ and $\mathcal{F}_1$ denote the event that $\mathcal{A}_I$ replaces the public key of entity $\mathtt{ID}_I$ at some point in its attack.

If $c = 0$, then $\mathcal{C}$ is a Type I challenger for `HybridPub` and begins by supplying $\mathcal{B}$ with a public key $K_{pub} = \langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, X, Y, Q, H_2, H_3, H_4 \rangle$. If $c = 1$, then $\mathcal{C}$ is a Type II challenger and so supplies $\mathcal{B}$ with a public key K_{pub} together with the value s such that $P_0 = sP$. Then $\mathcal{B}$ simulates the algorithm `Setup` of `FullCL-PKE` for $\mathcal{A}_I$ by supplying $\mathcal{A}_I$ with `params`$=$ $\langle \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2, H_3, H_4 \rangle$. Here H_1 is a random oracle that will be controlled by $\mathcal{B}$. Adversary $\mathcal{A}_I$ may make queries of the random oracles H_i, $1 \leq i \leq 4$, at any time during its attack. These are handled as follows:

H_1 **queries:** $\mathcal{B}$ maintains a list of tuples $\langle \mathtt{ID}_i, Q_i, b_i, x_i, X_i, Y_i \rangle$ which we call the H_1 list. The list is initially empty, and when $\mathcal{A}_I$ queries H_1 on input $\mathtt{ID} \in \{0,1\}^*$, $\mathcal{B}$ responds as follows:
(1) If `ID` already appears on the H_1 list in a tuple $\langle \mathtt{ID}_i, Q_i, b_i, x_i, X_i, Y_i \rangle$, then $\mathcal{B}$ responds with $H_1(\mathtt{ID}) = Q_i \in \mathbb{G}_1^*$.
(2) If `ID` does not already appear on the list and `ID` is the I-th distinct H_1 query made by $\mathcal{A}_I$, then $\mathcal{B}$ picks b_I at random from $\mathbb{Z}_q^*$, outputs $H(\mathtt{ID}) = b_I Q$ and

adds the entry $\langle \texttt{ID}, b_I Q, b_I, \perp, X, Y\rangle$ to the H_1 list.
(3) Otherwise, when ID does not already appear on the list and ID is the i-th distinct H_1 query made by $\mathcal{A}_I$ where $i \neq I$, $\mathcal{B}$ picks b_i and x_i at random from $\mathbb{Z}_q^*$, outputs $H(\texttt{ID}) = b_i P$ and adds $\langle \texttt{ID}, b_i P, b_i, x_i, x_i P, x_i P_0\rangle$ to the H_1 list.

Notice that with this specification of H_1, the FullCL-PKE partial private key for $\texttt{ID}_i$ ($i \neq I$) is equal to $b_i P_0$ while the public key for $\texttt{ID}_i$ is $\langle x_i P, x_i P_0\rangle$ and the private key for $\texttt{ID}_i$ is $x_i b_i P_0$. These can all be computed by $\mathcal{B}$ when $c = 0$. Additionally, when $c = 1$ (so $\mathcal{B}$ has s), $\mathcal{B}$ can compute $s b_I Q$, the partial private key of $\texttt{ID}_I$.

H_2 **queries:** Any H_2 queries made by $\mathcal{A}_I$ are passed to $\mathcal{C}$ to answer. We do need to assume in the course of the proof that H_2 is a random oracle.

H_3 **and** H_4 **queries:** Adversary $\mathcal{B}$ passes $\mathcal{A}_I$'s H_3 and H_4 queries to $\mathcal{C}$ to answer, but keeps lists $\langle \sigma_j, M_j, H_{3,j}\rangle$ and $\langle \sigma'_i, H_{4,i}\rangle$ of $\mathcal{A}_I$'s distinct queries and $\mathcal{C}$'s replies to them.

Phase 1: After receiving params from $\mathcal{B}$, $\mathcal{A}_I$ launches Phase 1 of its attack, by making a series of requests, each of which is either a partial private key extraction for an entity, a private key extraction for an entity, a request for a public key for an entity, a replacement of a public key for an entity or a decryption query for an entity. We assume that $\mathcal{A}_I$ always makes the appropriate H_1 query on the identity ID for that entity before making one of these requests. $\mathcal{B}$ replies to these requests as follows:

Partial Private Key Extraction: Suppose the request is on $\texttt{ID}_i$. There are three cases: (1) If $i \neq I$, then $\mathcal{B}$ replies with $b_i P_0$. (2) If $i = I$ and $c = 0$, then $\mathcal{B}$ aborts. (3) If $i = I$ and $c = 1$, then $\mathcal{B}$ replies with $s b_I Q$.

Private Key Extraction: Suppose the request is on $\texttt{ID}_i$. We can assume that the public key for $\texttt{ID}_i$ has not been replaced. There are two cases: (1) If $i \neq I$, then $\mathcal{B}$ outputs $x_i b_i P_0$. (2) If $i = I$, then $\mathcal{B}$ aborts.

Request for Public Key: If the request is on $\texttt{ID}_i$ then $\mathcal{B}$ returns $\langle X_i, Y_i\rangle$ by accessing the H_1 list.

Replace Public Key: Suppose the request is to replace the public key for $\texttt{ID}_i$ with value $\langle X'_i, Y'_i\rangle$. (We know that this will be a valid public key, i.e. a key satisfying $e(X'_i, P_0) = e(Y'_i, P)$). There are two cases: (1) If $i = I$ and $c = 1$, then $\mathcal{B}$ aborts. (2) Otherwise, $\mathcal{B}$ replaces the current entries X_i, Y_i in the H_1 list with the new entries X'_i, Y'_i. If $i = I$, then $\mathcal{B}$ makes a request to its challenger $\mathcal{C}$ to replace the public key components $\langle X, Y\rangle$ in K_{pub} with new values $\langle X'_I, Y'_I\rangle$.

Decryption Queries: Suppose the request is to decrypt ciphertext $\langle U, V, W\rangle$ for $\texttt{ID}_\ell$, where (as discussed in Section 3), the private key that should be used is the one corresponding to the current value of the public key for $\texttt{ID}_i$. Notice that even when $\ell = I$, $\mathcal{B}$ cannot make use of $\mathcal{C}$ to answer the query, because $\mathcal{B}$ is meant to be an IND-CPA adversary. Instead $\mathcal{B}$ makes use of an algorithm $\mathcal{KE}$.

Algorithm $\mathcal{KE}$**:** The input to the algorithm is a ciphertext $C = \langle U, V, W\rangle$, an identity $\texttt{ID}_\ell$ and the current value of the public key $\langle X_\ell, Y_\ell\rangle$. We assume that

$\mathcal{KE}$ also has access to the H_3 and H_4 lists. $\mathcal{KE}$ operates as follows:

(1) Find all triples $\langle \sigma_j, M_j, H_{3,j} \rangle$ on the H_3 list such that

$$\langle U, V \rangle = \texttt{BasicCL-PKE-Encrypt}_{\texttt{ID}_\ell, \langle X_\ell, Y_\ell \rangle}(\sigma_j; H_{3,j}).$$

Here, $\texttt{BasicCL-PKE-Encrypt}_{\texttt{ID}_A, \langle X_A, Y_A \rangle}(M; r)$ denotes the BasicCL-PKE encryption of message M for $\texttt{ID}_A$ using public key $\langle X_A, Y_A \rangle$ and random value r. Collect all these triples in a list S_1. If S_1 is empty, output $\perp$ and halt.
(2) For each triple $\langle \sigma_j, M_j, H_{3,j} \rangle$ in S_1, find all pairs $\langle \sigma'_i, H_{4,i} \rangle$ in the H_4 list with $\sigma_j = \sigma'_i$. For each such match, place $\langle \sigma_j, M_j, H_{3,j}, H_{4,i} \rangle$ on a list S_2. If S_2 is empty, then output $\perp$ and halt.
(3) Check in S_2 for an entry such that $W = M_j \oplus H_{4,i}$. If such an entry exists, then output M_j as the decryption of $\langle U, V, W \rangle$. Otherwise, output $\perp$.

We prove that $\mathcal{KE}$ correctly decrypts with high probability in Lemma 9.

Challenge Phase: At some point, $\mathcal{A}_I$ should decide to end Phase 1 and pick $\texttt{ID}_{\text{ch}}$ and two messages m_0, m_1 on which it wishes to be challenged. We can assume that $\texttt{ID}_{\text{ch}}$ has already been queried of H_1 but that $\mathcal{A}_I$ has not extracted the private key for this identity. Algorithm $\mathcal{B}$ responds as follows. If $\texttt{ID}_{\text{ch}} \neq \texttt{ID}_I$ then $\mathcal{B}$ aborts. Otherwise $\texttt{ID}_{\text{ch}} = \texttt{ID}_I$ and $\mathcal{B}$ gives $\mathcal{C}$ the pair m_0, m_1 as the messages on which it wishes to be challenged. $\mathcal{C}$ responds with the challenge ciphertext $C' = \langle U', V', W' \rangle$, such that C' is the HybridPub encryption of m_b under K_{pub} for a random $b \in \{0, 1\}$. Then $\mathcal{B}$ sets $C^* = \langle b_I^{-1} U', V', W' \rangle$ and delivers C^* to $\mathcal{A}_I$. It is easy to see that C^* is the FullCL-PKE encryption of m_b for identity $\texttt{ID}_I$ under public key $\langle X_I, Y_I \rangle$. We let $\langle X_{\text{ch}}, Y_{\text{ch}} \rangle$ denote the particular value of the public key for identity $\texttt{ID}_{\text{ch}}$ during the challenge phase ($\mathcal{A}_I$ may change this key in Phase 2 of its attack).

Phase 2: $\mathcal{B}$ continues to respond to $\mathcal{A}_I$'s requests as in Phase 1.

Guess: Eventually, $\mathcal{A}_I$ should make a guess b' for b. Then $\mathcal{B}$ outputs b' as its guess for b. If $\mathcal{A}_I$ has used more than time t, or attempts to make more than q_i queries to random oracle H_i or more than q_d decryption queries, then $\mathcal{B}$ should abort $\mathcal{A}_I$ and output a random guess for bit b (in this case algorithm $\mathcal{KE}$ has failed to perform correctly at some point).

Analysis: We claim that if algorithm $\mathcal{B}$ does not abort during the simulation and if all of $\mathcal{B}$'s uses of the algorithm $\mathcal{KE}$ result in correct decryptions, then algorithm $\mathcal{A}_I$'s view is identical to its view in the real attack. Moreover, if this is the case, then $2(\Pr[b = b'] - \frac{1}{2}) \geq \epsilon$. This is not hard to see: $\mathcal{B}$'s responses to all hash queries are uniformly and independently distributed as in the real attack. All responses to $\mathcal{A}_I$'s requests are valid, provided of course that $\mathcal{B}$ does not abort and that $\mathcal{KE}$ performs correctly. Furthermore, the challenge ciphertext C^* is a valid FullCL-PKE encryption of m_b under the current public key for identity $\texttt{ID}_{\text{ch}}$, where $b \in \{0, 1\}$ is random. Thus, by definition of algorithm $\mathcal{A}_I$ we have that $2(\Pr[b = b'] - \frac{1}{2}) \geq \epsilon$.

So we must examine the probability that $\mathcal{B}$ does not abort during the simulation given that the algorithm $\mathcal{KE}$ performs correctly. Examining the simulation, we see that $\mathcal{B}$ can abort for one of four reasons:

(0) Because $c = 0$ and the event $\mathcal{F}_0$ occurred during the simulation.
(1) Because $c = 1$ and event $\mathcal{F}_1$ occurred during the simulation.
(2) Because $\mathcal{A}_I$ made a private key extraction on $\mathtt{ID}_I$ at some point.
(3) Or because $\mathcal{A}_I$ chose $\mathtt{ID}_{\rm ch} \neq \mathtt{ID}_I$.

We name the event $(c = i) \wedge \mathcal{F}_i$ as $\mathcal{H}_i$ for $i = 0, 1$. We also name the last two events here as $\mathcal{F}_2$ and $\mathcal{F}_3$. Of course, $\mathcal{F}_3$ is the same as event $\neg\mathcal{H}$. Now $\mathcal{A}_I$ makes q_1 queries of H_1 and chooses $\mathtt{ID}_{\rm ch}$ from amongst the responses $\mathtt{ID}_i$, while $\mathcal{B}$'s choice of I is made uniformly at random from the set of q_1 indices i. So the probability that $\mathtt{ID}_{\rm ch} = \mathtt{ID}_I$ is equal to $1/q_1$. Hence $\Pr[\mathcal{H}] = 1/q_1$. Notice too that the event $\neg\mathcal{F}_3$ implies the event $\neg\mathcal{F}_2$ (if $\mathcal{A}_I$ chooses $\mathtt{ID}_{\rm ch} = \mathtt{ID}_I$, then no private key extraction on $\mathtt{ID}_I$ is allowed). Gathering this information together:

$$\Pr[\mathcal{B} \text{ does not abort}] = \Pr[\neg\mathcal{H}_0 \wedge \neg\mathcal{H}_1 \wedge \neg\mathcal{F}_2 \wedge \neg\mathcal{F}_3] = \frac{1}{q_1} \cdot \Pr[\neg\mathcal{H}_0 \wedge \neg\mathcal{H}_1 | \mathcal{H}].$$

Notice now that the events $\mathcal{H}_0$ and $\mathcal{H}_1$ are mutually exclusive (because one involves $c = 0$ and the other $c = 1$). Therefore we have

$$\Pr[\neg\mathcal{H}_0 \wedge \neg\mathcal{H}_1 | \mathcal{H}] = 1 - \Pr[\mathcal{H}_0 | \mathcal{H}] - \Pr[\mathcal{H}_1 | \mathcal{H}].$$

Moreover, $\Pr[\mathcal{H}_i | \mathcal{H}] = \Pr[(c = i) \wedge \mathcal{F}_i | \mathcal{H}] = \frac{1}{2}\Pr[\mathcal{F}_i | \mathcal{H}]$, where the last equality follows because the event $\mathcal{F}_i | \mathcal{H}$ is independent of the event $c = i$. So we have

$$\Pr[\mathcal{B} \text{ does not abort}] = \frac{1}{q_1}\left(1 - \frac{1}{2}\Pr[\mathcal{F}_0 | \mathcal{H}] - \frac{1}{2}\Pr[\mathcal{F}_1 | \mathcal{H}]\right).$$

Finally, we have that $\Pr[\mathcal{F}_0 \wedge \mathcal{F}_1 | \mathcal{H}] = 0$ because of the rules on adversary behaviour described in Section 3 (an adversary cannot both extract the partial private key and change the public key of the challenge identity). This implies that $\Pr[\mathcal{F}_0 | \mathcal{H}] + \Pr[\mathcal{F}_1 | \mathcal{H}] \leq 1$. Hence we see that $\Pr[\mathcal{B} \text{ does not abort}] \geq 1/2q_1$.

Now we examine the probability that algorithm $\mathcal{KE}$ correctly handles all of $\mathcal{A}_I$'s q_d decryption queries. We will show in Lemma 9 below that the probability that $\mathcal{KE}$ correctly replies to individual decryption queries is at least λ, where λ is bounded as in the statement of this lemma.

It is now easy to see that $\mathcal{B}$'s advantage is at least $\frac{\epsilon}{2q_1}\lambda^{q_d}$. It follows that either $\mathcal{B}$'s advantage as a Type I adversary against `HybridPub` or $\mathcal{B}$'s advantage as a Type II adversary against `HybridPub` is at least $\frac{\epsilon}{4q_1}\lambda^{q_d}$. The running time of $\mathcal{B}$ is $\text{time}(\mathcal{A}_I) + q_d \cdot \text{time}(\mathcal{KE}) = t + O((q_3 + q_4)q_d t')$ where t' is the running time of the `BasicCL-PKE` encryption algorithm. This completes the proof.

Lemma 9 *In the simulation in the proof of Lemma 2, Algorithm $\mathcal{KE}$ correctly replies to individual decryption queries with probability at least λ where*

$$\begin{aligned} 1 - \lambda \quad \leq \quad & (q_3 + q_4) \cdot \epsilon_{\scriptscriptstyle OWE}(t + O((q_3 + q_4)q_d t', q_2) \\ & + \epsilon_{\scriptscriptstyle GBDHP}(t + O((q_3 + q_4)q_d t') + 3q^{-1} + 2^{-n+1}. \end{aligned}$$

Here t is the running time of adversary $\mathcal{A}_I$, t' is the running time of the `BasicCL-PKE` *encryption algorithm, $\epsilon_{\scriptscriptstyle OWE}(T, q')$ denotes the highest advantage of any Type I or Type II OWE adversary against* `BasicPub` *which operates in time T and makes q' hash queries to H_2, and $\epsilon_{\scriptscriptstyle GBDHP}(T)$ denotes the highest advantage of any algorithm to solve GBDHP in time T in groups of order q generated by $\mathcal{IG}$.*

Proof of Lemma 9: The proof, which is given in [2], is closely modelled on the proof of [11, Lemma 11], but differs in several key respects: we need to build an algorithm which handles multiple public keys, and the algorithm can be asked to decrypt the challenge ciphertext (but under a different identity/public key combination from the challenge identity). This substantially complicates the analysis.

Proof of Lemma 3: This proof is modelled on the proof of [11, Lemma 10], modified to handle $\mathcal{A}_I$'s ability to replace public keys. See [2] for details.

Proof of Lemma 4: The proof technique is similar to that used in Lemma 3.

Proof of Lemma 5: This proof is similar to that of [5, Theorem 4.1], with modifications to handle adversaries who can replace public keys.

Proof of Lemma 6: The proof is in the full version [2]; it uses ideas from both the $c = 1$ case of the proof of Lemma 2, and the proof of [5, Lemma 4.6].

Proof of Lemma 7: This is easily proven using [11, Theorem 14], noting that s can be made available to Type II adversaries simply by including it in public keys. We also use the fact that `HybridPub` is $1/q$-uniform in the sense of [11].

Proof of Lemma 8: The proof in [2] uses similar techniques to the proof of Lemma 5 with a twist to handle the Type II adversary's knowledge of s.