Escrow-Free Encryption Supporting Cryptographic Workflow

S.S. Al-Riyami[1], J. Malone-Lee[2] and N.P. Smart[2]

[1] Information Security Group,
Royal Holloway, University of London,
Egham,
Surrey, TW20 0EX,
United Kingdom.
sattam@gmail.com

[2] Dept. Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
{malone,nigel}@cs.bris.ac.uk

Abstract. Since Boneh and Franklin published their seminal paper on identity based encryption (IBE) using the Weil pairing , there has been a great deal of interest in cryptographic primitives based on elliptic-curve pairings. One particularly interesting application has been to control access to data, via possibly complex policies. In this paper we continue the research in this vein. We present an encryption scheme such that the receiver of an encrypted message can only decrypt if it satisfies a particular policy chosen by the sender at the time of encryption. Unlike standard IBE, our encryption scheme is escrow free in that no key-issuing authority (or colluding set of key-issuing authorities) is able to decrypt ciphertexts itself. In addition we describe a security model for the scenario in question and provide proofs of security for our scheme (in the random oracle model). [1]

1 Introduction

Suppose that Alice wants to send a message to Bob. She wants to be able to encrypt the message in such a way that Bob can only decrypt if he has a particular set of *authorisation credentials* (or simply *credentials*). Alice specifies the credentials that Bob should have in a *policy* that she decides before encrypting.

[1] The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability

Alice should be able to perform this encryption without knowing what credentials Bob actually has. The possession of credentials is validated by *authorisation authorities*. If validation is successful then these authorities issue *authorisation certificates* that act as (partial) decryption keys. Alice wants to be sure that no colluding set of these authorisation authorities is able to decrypt and recover the message that she intended for Bob. In other words, the system should be *escrow free*.

In this paper we describe a framework in which to solve the problem outlined above and we give a corresponding security model. We present a solution to the problem and provide proofs of security. Our solution uses ideas originating from Boneh and Franklin identity based encryption (IBE) scheme [4]. One of the main benefits of IBE is that encryption can occur at any time, even before the private key of the recipient has been created. This temporal difference in the possible order of encryption and key generation lead Paterson to coin the phrase *cryptographic workflow* [12]. The expression workflow is used to describe a system in which actions must be performed in a particular order: you must do X before you are able to do Y. Or, as a more concrete example, A could encrypt a cheque to B but only give B the decryption key once B has performed some task. Our system supports this concept of workflow in that recipients of encrypted messages do not have to have access to the requisite credentials for decryption at the time of encryption. This means that authorisation authorities can ensure that some action has been performed, or that some event has occurred, before issuing credentials.

Several related questions have already been considered in the literature. In [11] the problem of obtaining public key certificates as part of an *automated trust negotiation* is considered. In this scenario one wishes to exchange certificates with someone, but only if they already hold a given set of other certificates. However, the possession of certificates may be confidential information, hence a method needs to be found to break the possible deadlock that could occur in such negotiations. A solution to this problem called *oblivious signature-based envelope* (OSBE) as described in [11]. This idea was extended to other *hidden credentials* (other than possession of certificates) in [6, 10].

The problem that we wish to solve here is a form of access control: access to a message contained in a ciphertext. In [7, 16] access control mechanisms are considered based on elliptic–curve pairings. The motivation is to minimise the number of pairing computations needed to evaluate the credentials in an access-policy. This leads to complicated schemes that do not allow the expression of all possible policies. In addition, the resulting schemes come with no proof of security.

In the system that we propose, and all those that we have mentioned above, there are authorities that issue some form of certificate or key necessary for decryption – subject to possession of some credential. The principal shortcoming with all the systems proposed to date is that, unless a secure channel is already available, they suffer from an escrow property: it is not only the legitimate holder of the credentials specified in an encryptor's policy that can decrypt, but also any collusion of authorities who are able to issue sufficient credentials needed to satisfy the policy. In addition any adversary that is able to obtain such, can also decrypt. In some applications, such as disaster recovery, this escrow facility may be useful. However, for most other applications this escrow facility is undesirable.

Key escrow is an inherent property of IBE since it is necessary to have a *trusted authority* to compute private keys. It is from IBE that the existing schemes in the literature inherit the escrow property [6, 7, 10, 11, 16]. In [2] Al-Riyami and Paterson describe a modification of the Boneh–Franklin IBE scheme which avoids the problem of key escrow in standard IBE. This method, called *certificateless public key cryptography* (CL-PKC), requires each user to have a (possibly unauthenticated) public key. Messages are then encrypted using a combination of a user's public key and its identity based key. Our system uses a similar idea to that of CL-PKC to avoid key escrow.

2 Cryptographic Workflow and Privacy Protection

In this section we give a more concrete description of the various entities involved in our solution.

Certification Authorities : Each user who wishes to receive encrypted messages must have a public key. There is a certification authority, or possibly more than one, which issues standard public key certificates. The certificates issued by this authority binds the identity of a user with the user's public key. The certification authority validates public keys and may ensure that parties actually know the private key corresponding to their public key via a proof of possession.

Authorisation Authorities : These are authorities which issue authorisation certificates that act as partial decryption keys for users receiving encrypted messages. These certificates are issued subject to possession of a particular credential.

A credential is described by a string, say s. In our scheme authorisation certificates take the form of digital signatures. For example the authorisation certificate issued by authority i for credential described by string s is the signature of the authority on s. We denote this $C_i(s)$. To avoid a very cumbersome description, henceforth we refer to authorisation certificates $C_i(s)$ as credentials.

Let us consider some example credentials. Suppose that the string s contains a copy of user Bob's public key, denoted by PK. In this case credential $C_i(s)$ binds the remainder of string s to PK, like an authorisation certificate in SPKI. In this case one does not need to transfer the credential from the authorisation authority to Bob in a confidential manner. See [1] or [2] for more details on this latter point.

On the other hand the string s may not need to be bound to a public key (or identity) and could simply be publicly broadcast. Since s is not unique for a specific decryptor or ciphertext, $C_i(s)$ can be used by any decryptor. Such a situation can be envisaged when the authority i is a time-stamping authority and the string s is simply a time value, the single credential being issued at the time point given by the string s. Hence, possession of such a credential $C_i(s)$ implies that the current time is greater than the time given by s. This allows encryption to an arbitrary point in the future.

Encryption/Decryption : In our framework anyone can encrypt a message to a user with a certified public key. At encryption time the encryptor can choose an arbitrary access structure that depends on possession of certain authorisation credentials $C_i(s)$. Such a structure will be described in a policy. The policy can be different for each message. This encryption and the associated decryption satisfies the following properties:

1. The encryption can be performed before the credentials $C_i(s)$ are computed. (At the time of encryption credential $C_i(s)$ can be specified by (i, s) where i is an authority and s is a string.) This property enables workflow.
2. No entity (including any colluding set of authorisation authorities) bar the recipient can learn anything about the encrypted message. This property guarantees the escrow free nature of the system. In addition if the encryptor does not require the escrow-free property, for example he may wish to broadcast the data to a number of recipients and not just a single entity, then our system can easily accomodate this feature, see Appendix A.2 for a further discussion of this feature.
3. The recipient cannot learn anything about the received message until it has obtained an appropriate set of authorisation credentials – as specified by the policy used for encryption. Such a set will be referred to as a *qualifying set* henceforth.

We now provide an example to demonstrate the powerful versatility of our scheme to combine access structures with cryptographic workflow.

A service provider (Alice) provides users (a particular user Bob in this case) with credentials that can be redeemed by any licensed online benefits agency. For example we consider the case of Bob being a pensioner claiming, possibly means-tested, benefits. A credential will confirm that Bob is allowed to obtain a particular benefit and that he is eligible for an extra payment since he lies within a particular social group (say D). For example, one valid set of credentials that Bob can satisfy before obtaining a credential could contain the strings: $s_1 = \mathsf{Group.D}\|PK\|\mathsf{2004}$, $s_2 = \mathsf{over.65.years.old}\|PK$, $s_3 = \mathsf{pension.X}\|nonce$. Alternatively, Bob can satisfy the singular credential that contains a string $s_4 = \mathsf{benefits.clerk}\|PK\|\mathsf{2004}$. One can imagine a situation in which Bob can obtain his pension if he holds either the set of credentials $\{C_1(s_1), C_1(s_2), C_2(s_3)\}$ or the single credential $C_3(s_4)$, where the first authority is the benefits agency and the second one is an online benefits website whilst the third authority is a physical benefits clark.

The string PK denotes Bob's public key and *nonce* is a random string of suitable length. Bob could already have the appropriate credential $C_1(s_1)$ documenting his membership of social group D and $C_1(s_2)$ stating that he is over 65 years old. Unlike the above two credentials, $C_2(s_3)$ can only be obtained after the ciphertext is produced (due to the nonce), this credential can be provided by an online benefit clerk say.

Credentials obtained from Alice can regulate the conduct of payment offices and allow citizens to qualify for extra payments etc. If a patient's representative is allowed to redeem the credential (as with U.K. pensions), additional privacy can be provided to the citizen since the payment agency will not deal directly with the citizen. This is particularly relevant since some people consider there to be a stigma associated with claiming means-tested benefits, especially amongst the older population.

Now let us examine these credentials. The nonce in s_3 allows Alice to ensure that $\mathsf{pension.X}$ is fresh and can be obtained at *any* time after the ciphertext is produced. Here s_3 is unique for each ciphertext or transaction. Additionally, the credential encrypted using the string $\mathsf{pension.X}$ will have a one-time use, provided that payment offices share a list of redeemed credentials. Furthermore, even if $C_2(s_3)$ is somehow revealed, it cannot be used (except by Alice due to the nonce) to provide or infer any information (e.g. whether Bob is a member of a particular social group), since it is anonymous and does not contain identifying information. Anonymous credentials can also be un-linkable by Alice if nonces are not used. Notice that both $C_1(s_1)$ and $C_1(s_2)$ are unique for the decryptor and could be publicly disclosed (due to the certified PK). Of course both credentials could be confidentially kept if disclosure is unnecessary or if they are considered sensitive. The credential $C_i(s_1)$ can be updated to $C_i(\mathsf{Group.D}\|PK\|\mathsf{2005})$ using only public channels and s_2 does not need to be updated or revoked. To make the policy less exclusive credentials such as $C_3(s_4)$ are included.

The trust established between Alice and Bob is easy to deploy without violating Bob's privacy. Assuming credentials (under very exclusive policies) are not revealed, the single round nature of the interaction allows Alice to gather no information on Bob. The above example can be further refined to preserve more of the Bob's privacy if the certificate from the certification authority uses a pseudonym as an identity and/or it is possible to obtain $C_i(s)$ totally anonymously from the authorisations authorities (here s cannot contain PK). Now even a collusion of authorisation authorities cannot construct any information for any entity.

3 Building Blocks

3.1 Groups With Pairings

We assume three finite abelian groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, all of order divisible by some large prime q and such that the discrete logarithm problem in each of the three groups is intractable. We assume that $\mathbb{G}_1$ and $\mathbb{G}_2$ are isomorphic as groups with a computable (but not necessarily invertible) isomorphism given by

$$\phi : \mathbb{G}_2 \longrightarrow \mathbb{G}_1.$$

We also assume that there is a computable bilinear pairing

$$\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T.$$

In practise $\mathbb{G}_1$ and $\mathbb{G}_2$ will be related to the (additive) group of points on an elliptic curve and $\mathbb{G}_T$ will be a subgroup of the (multiplicative) group of a finite field. Hence, we use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$ and multiplicative notation for $\mathbb{G}_T$. Note, in [2] it is assumed that $\mathbb{G}_1 = \mathbb{G}_2$, the generalisation to the case where $\mathbb{G}_1 \neq \mathbb{G}_2$ requires some additional complications below which the reader will be able to ascertain without further comment.

Not only do we wish the discrete logarithm problem in the three groups to be intractable, but we also require the following problem to also be intractable.

Definition 1 (Bilinear-Diffie–Hellman problem (BDHP)). *Given $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ as above, with P' and $P = \phi(P')$ the generators of $\mathbb{G}_2$ and $\mathbb{G}_1$ respectively, the BDHP problem is as follows. Given $\{[x]P, [y]P', [z]P\}$, for $x, y, z \in \mathbb{F}_q^*$ chosen uniformly at random, compute $\hat{t}(P, P')^{xyz}$.*

The advantage of a polynomially bounded adversary $\mathcal{A}$ in solving the BDHP is defined by

$$\mathrm{Adv}_{\mathrm{BDHP}}(\mathcal{A}) = \Pr\left[\mathcal{A}([x]P, [y]P', [z]P) = \hat{t}(P, P')^{xyz}\right].$$

We let $P' \in \mathbb{G}_2$ denote a fixed base point in $\mathbb{G}_2$ of order q, and $P = \phi(P') \in \mathbb{G}_1$ a fixed base point of order q in $\mathbb{G}_1$.

3.2 Secret Sharing Scheme

Given m terms, an access structure, or policy, is a boolean expression involving only conjunction and disjunction of the m terms. We denote this policy $\mathcal{P}$. In our final scheme each term will represent possession of some credential. For example one may wish to allow access to the data on possession of the credential C_1 or possession of both C_2 and C_3, in which case the policy is given by

$$C_1 \vee (C_2 \wedge C_3).$$

We assume a secret sharing scheme generating algorithm $\mathbb{S}_m$ which, on input of an access structure/policy $\mathcal{P}$ involving at most m terms and a message/secret $M \in \{0,1\}^{l_s}$, will output a set of shares $\mathbb{K} = \{\mathfrak{b}_j\}_{j=1}^m$, with $\mathfrak{b}_j \in \{0,1\}^l$ and public auxiliary information A,

$$(\mathbb{K}, A) \longleftarrow \mathbb{S}_m(\mathcal{P}, M).$$

We insist that there is one share $\mathfrak{b}_j$ for each of the m terms/participants in the access structure $\mathcal{P}$. In addition we assume a combining algorithm which on input of a subset of shares $\mathbb{K}' \subset \mathbb{K}$ will output

$$\mathbb{S}_m^{-1}(\mathbb{K}', A) = \begin{cases} M & \text{If } \mathbb{K}' \text{ is a qualifying subset of } \mathbb{K}, \\ \perp & \text{Otherwise.} \end{cases}$$

Note, we do not assume that the inverse algorithm requires as input the policy $\mathcal{P}$; however, if in a particular instance it does, we assume that $\mathcal{P}$ is part of the auxiliary information A.

The usual definition of security for a secret sharing scheme is Definition 2.

Definition 2. *A secret sharing scheme generator $\mathbb{S}_m$ is called perfect if for all access structures/policies $\mathcal{P}$*

1. *Given a qualifying subset $\mathbb{K}'$, one can recover the whole secret.*
2. *Given only a non-qualifying subset $\mathbb{K}'$, one can recover no information about the shared secret.*

However, we can restrict to a computationally bounded notion of security. Such a scheme we shall call semantically secure if no polynomially bounded adversary can win the following game with non-negligible probability in the security parameter l. We let m be fixed. In the first part of the game the adversary chooses a non-trivial access structure/policy $\mathcal{P}$ on m symbols and two messages M_0 and M_1. The adversary $\mathcal{A}$ passes $(\mathcal{P}, M_0, M_1)$ to the challenger who picks a bit b and runs the secret sharing scheme generator

$$(\mathbb{K}, A) \longleftarrow \mathbb{S}_m(\mathcal{P}, M_b).$$

The challenger passes A back to the adversary, who then eventually terminates with his guess b' as to the hidden bit b. During the second stage of the game the adversary is allowed to adaptively ask the challenger for shares $\mathfrak{b}_j \in \mathbb{K}$ of its choice. This is subject to the condition that the adversary must not ask for a subset of shares which is a qualifying set.

The advantage of the adversary is defined to be

$$\mathrm{Adv}_{\mathbb{S}_m}(\mathcal{A}) = 2 \cdot \left|\Pr[b' = b] - \frac{1}{2}\right| = \left|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]\right|.$$

As an example, let us consider how the above model fits the security of a threshold scheme with threshold value t. In this case we are saying that an adversary cannot determine, given the appropriate auxiliary information and up to $t-1$ shares, which of two secrete has been shared.

Note, introducing a computational notion of security for secret sharing schemes and using it in our security proofs instead of the more traditional information theoretic definition means our security proofs are stronger, as they rely on weaker assumptions. However, we envisage that one would in fact use one of the information theoretically secure secret sharing schemes such as that by Shamir [14] (in the case of threshold schemes) or Benaloh and Leichter [3] (in the case of general access structures).

3.3 Symmetric Encryption Function

Let Enc denote a symmetric encryption function with key space $\{0,1\}^{l_e}$. We adopt the security notion of find-guess [9] for such schemes in this paper and assume that all symmetric encryption functions are secure in the following sense. We denote find-guess security by FG security henceforth.

Let $\mathcal{A}$ be an polynomially bounded adversary against Enc in the sense of FG. The algorithm $\mathcal{A}$ runs in two stages. In the first stage the adversary selects two distinct messages M_0 and M_1 of equal length. These messages are passed to a challenger who selects a bit b and a symmetric key $k \in \{0,1\}^{l_e}$. The challenger computes $C^* = \mathrm{Enc}_\kappa(M_b)$ and passes this back to the adversary $\mathcal{A}$. The second stage the adversary completes by $\mathcal{A}$ returning its guess b' as to the hidden bit b. The advantage of $\mathcal{A}$ is defined to be

$$\mathrm{Adv}_{\mathrm{Enc}}(\mathcal{A}) = 2 \cdot \left| \Pr[b' = b] - \frac{1}{2} \right| = \left| \Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0] \right| .$$

We say that Enc is secure in the sense of FG if no such adversary exists with advantage greater than a negligible function of l_e. Note, the adversary has no oracle access to an encryption or decryption oracle in this definition.

3.4 Hash Functions

Define the following hash functions, which we assume are modelled by random oracles,

$$\begin{aligned} H_1 &: \{0,1\}^* \longrightarrow \mathbb{G}_2, \\ H_2 &: \mathbb{G}_T \longrightarrow \{0,1\}^l, \\ H_3 &: \{0,1\}^* \longrightarrow \mathbb{F}_q^*, \text{ and} \\ H_4 &: \{0,1\}^{l_s} \longrightarrow \{0,1\}^{l_e}. \end{aligned}$$

where $l \approx \log_2 q$, l_e is the key size of the symmetric encryption function and l_s is the size of the input to the secret sharing scheme.

4 Scheme Definition

In this section we describe how our scheme is constructed from the building blocks introduced in Section 3.

Authorisation Authority Setup :

Each authority $i \in \{1, \ldots, n\}$, generates it own private key by choosing $\mathfrak{a}_i \in \mathbb{F}_q^*$ uniformly at random. It computes its public key as

$$R_i = [\mathfrak{a}_i]P.$$

A certified version of $R_i \in \mathbb{G}_1$ is given to all users. Note that this is the same way that keys are generated for the BLS signature scheme [5].

User Setup :

Each user (say Bob) generates its own private key by choosing $\mathfrak{u} \in \mathbb{F}_q^*$ uniformly at random. It computes its public key as

$$PK = (X, Q) \in \mathbb{G}_1 \times \mathbb{G}_2,$$

where $X = [\mathfrak{u}]P \in \mathbb{G}_1$, and $Q = H_1(X) \in \mathbb{G}_2$, where to apply the hash function we interpret X as a bit string with some given convention.

The public keys of users are made available to other users in a certified manner, via a certificate issued by a certification authority (as discussed in Section 2).

Credential Issuing (by Authorisation Authority i):

Suppose Bob wishes to obtain the credential $C_i(s)$. Bob obtains the element of $\mathbb{G}_2$ given by

$$C_i(s) = [\mathfrak{a}_i]Q_s,$$

where $Q_s = H_1(s) \in \mathbb{G}_2$ from the ith authorisation authority.

Of course, before issuing $C_i(s)$, authority i verifies that Bob satisfies the requirements described by the string s.

Note, in CL-PKC [2] and IBE [4] this corresponds to the identity based partial private key extraction phase. In the language of OSBE [11] we are simply obtaining the credential $C_i(s)$ of authority i on the string s. It is this latter terminology which we shall adopt, since one can see that the authorisation authority is simply providing Bob with a BLS signature [5] on the string s.

Encryption :

First, the encryptor (Alice) decides what credentials the decryptor (Bob) should have. Let us denote these $\{C_{i_j}(s_j)\}_{j=1}^m$ where $m \leq n$ and n is the number of authorisation authorities and $i_j \in \{1, \ldots, n\}$. For $j = 1, \ldots, m$, these correspond to credentials $C_{i_j}(s_j)$ issued by authorisation authority i_j for string s_j. The string s_j describes what conditions should be satisfied before the credential

$$C_{i_j}(s_j) = [\mathfrak{a}_{i_j}]Q_{s_j} = [\mathfrak{a}_{i_j}]H_1(s_j)$$

is issued by authority i_j. The set $\{(i_j, s_j)\}_{j=1}^m$ is included in the policy $\mathcal{P}$ along with a conjunction and disjunction of m terms to describe the qualifying sets for decryption.

The encryptor then performs the following steps,

1. Generate $\mathfrak{b} \in \{0,1\}^{l_s}$ at random.
2. Perform the secret sharing to obtain

$$(\mathbb{K}, A) = \left(\{\mathfrak{b}_j\}_{j=1}^m, A\right) = \mathbb{S}_m(\mathcal{P}, \mathfrak{b}).$$

3. Set $E = \mathrm{Enc}_{H_4(\mathfrak{b})}(M)$.
4. Set $r = H_3(\mathfrak{b} \| \mathcal{P} \| M)$.
5. Compute $U = [r]P$.
6. Compute $g = \hat{t}([r]X, Q)$.
7. For all $1 \leq j \leq m$
   (a) Compute $Q_{s_j} = H_1(s_j) \in \mathbb{G}_2$.
   (b) Compute $g_j = \hat{t}([r]R_{i_j}, Q_{s_j})$.

(c) Compute $c_j = \mathfrak{b}_j \oplus H_2(g_j) \oplus H_2(g)$.
8. Return $(U, \{c_j\}_{j=1}^m, A, \mathcal{P}, E)$.

Informally, this provides us with an encryption E of the message M, under a symmetric key derived from $\mathfrak{b}$ using H_4, plus a set of shares $\mathbb{K} = \{\mathfrak{b}_j\}_{j=1}^m$. There is one share $\mathfrak{b}_j \in \{0,1\}^l$ corresponding to each (i_j, s_j) in the policy $\mathcal{P}$. There is also a mechanism to recover $\mathfrak{b}$ given the shares in $\mathbb{K}$. The shares required by the secret sharing scheme are then encrypted using an analogue of the ElGamal encryption mechanism. Chosen ciphertext security is provided by r being a function of $\mathfrak{b}$, $\mathcal{P}$ and the message M.

The escrow free nature of the encryption is provided by the generation of an "identity" Q depending on the recipients public key, which is decrypted by the user acting as their own trust authority.

Decryption :

Let $\mathcal{B}_C$ denote the set of credentials possessed by Bob on and let $\mathcal{P}_C$ denote the set of credentials implied by the policy $\mathcal{P}$.

1. Compute $g' = \hat{t}([\mathfrak{u}]U, Q)$.
2. For each such credential $C_i(s_j) \in \mathcal{B}_C \cap \mathcal{P}_C$ do:
   (a) Set $g'_j = \hat{t}(U, C_i(s_j))$.
   (b) Set $\mathfrak{b}'_j = c_j \oplus H_2(g'_j) \oplus H_2(g')$.
3. Let $\mathbb{K}'$ be the resulting set of $\mathfrak{b}'_j$ after step 1.
4. Compute $\mathfrak{b} = \mathbb{S}_m^{-1}(\mathbb{K}', A)$. If $\mathfrak{b} = \perp$ then return $\perp$.
5. Compute $M = \mathrm{Enc}_{H_4(\mathfrak{b})}^{-1}(E)$.
6. Compute $r' = H_3(\mathfrak{b} \| \mathcal{P} \| M)$.
7. If $U = [r']P$ then return M, otherwise return $\perp$.

Note that decryption works, and can only be performed by the user holding his private key $\mathfrak{u}$ since

$$\begin{aligned} g'_j &= \hat{t}(U, C_{i_j}(s_j)) = \hat{t}([r]P, [\mathfrak{a}_{i_j}]Q_{s_j}) \\ &= \hat{t}([r][\mathfrak{a}_{i_j}]P, Q_{s_j}) = \hat{t}([r]R_{i_j}, Q_{s_j}) \\ &= g_j \end{aligned}$$

and

$$\begin{aligned} g' &= \hat{t}([\mathfrak{u}]U, Q) = \hat{t}([r][\mathfrak{u}]P, Q) \\ &= \hat{t}([r]X, Q) \\ &= g. \end{aligned}$$

In addition the encryption is escrow free since only the intended participant knows the private key $\mathfrak{u}$.

5 Security Models and Results

We require two definitions of security for our scheme: one to guarantee the escrow freeness of the encryption scheme and one to ensure that a recipient cannot break the semantic security of the scheme without obtaining a qualifying subset of the credentials in a policy. We call these two properties *external security* and *recipient security* respectively.

The situation is somewhat similar to that in [2], but the details of the model are slightly different. In [2] two types of adversary are also defined: A Type I adversary does not have access to the private keys of the authority but is able to alter participants public keys, a Type II adversary does have access to the private keys of the authority but is unable to mount such a key substitution attack.

5.1 External Security: Definition and Result

In this scenario, what we would like to show is that no adversary is able to learn anything about an encrypted message, even when it knows the private keys of all the authorisation authorities. This models the escrow-free property that we aim to achieve. Following the convention that security against adaptive chosen ciphertext attack [13] is the correct notion of security for encryption schemes, we also grant an adversary access to a decryption oracle that it may call adaptively.

Our security definition is presented by the following game played by an adversary $\mathcal{A}$ against a challenger. The adversary takes as input the certified public key of a user Bob, plus the private keys of all n authorisation authorities. The adversary is assumed to run in the random oracle model and is therefore given oracle access only to the hash functions H_1, H_2, H_3 and H_4. The adversary also has access to a decryption oracle $\mathcal{D}$ (see Section A.1 for a discussion of this oracle). The adversary is assumed to be parametrised by a value $m \in \mathbb{N}$.

Stage 1: The adversary can adaptively query the decryption oracle $\mathcal{D}$ with ciphertexts of its choosing. The decryption oracle responds as it would if it was using Bob's private key.

At the end of Stage 1 the adversary outputs two equal length messages M_0 and M_1, a non-trivial access policy $\mathcal{P}^*$ on m terms and a state S.

Challenge: The simulator chooses a bit b and, using the encryption algorithm above, it encrypts M_b under Bob's public key and the policy $\mathcal{P}^*$. The resulting ciphertext C^* and the state S are passed back to the second stage of the adversary.

Stage 2: The adversary can adaptively query the decryption oracle $\mathcal{D}$ on ciphertexts C of his choosing subject to the restriction that it does not make a query on ciphertext C^* (or a related ciphertext as discussed in Appendix A.1).

Guess: At the end of Stage 2 the adversary outputs a bit b': its guess at b. The adversary is said to win the game if $b = b'$. We define the advantage of the adversary as

$$\mathrm{Adv}_{\mathrm{Ext}}(\mathcal{A}) = 2 \cdot \left|\Pr[b' = b] - \frac{1}{2}\right| = |\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]| \,.$$

Theorem 1. *Suppose that there is an adversary $\mathcal{A}$ of our scheme that makes at most q_i calls to random oracle i and that it makes at most q_d calls to $\mathcal{D}_{\mathcal{A}}$, then we have algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ that are of similar efficiency and such that*

$$\begin{aligned}\mathrm{Adv}_{\mathrm{Ext}}(\mathcal{A}) \leq\ & 2 \cdot q_2 \cdot \mathrm{Adv}_{\mathrm{BDHP}}(\mathcal{B}_1) + 2 \cdot \mathrm{Adv}_{\mathbb{S}_m}(\mathcal{B}_2) + 2 \cdot \mathrm{Adv}_{\mathrm{Enc}}(\mathcal{B}_3) \\ & + \frac{q_3(q_3 - 1) + 2q_d}{q - 1} + \frac{q_3 + q_4}{2^{l_s - 2}}.\end{aligned}$$

Proof. The proof can be found in Appendix B.

5.2 Recipient Security: Definition and Result

In this scenario, what we would like to show is that no adversary is able to learn anything about encrypted messages, even when it knows the private key of the recipient and is able to obtain all credentials in the target policy $\mathcal{P}^*$ (the policy chosen by the adversary for the generation of the challenge ciphertext in the game described below), subject to the restriction it is unable to obtain all credentials from any qualifying set in $\mathcal{P}^*$. We also grant an adversary additional power via oracle queries that we describe in the attack game below.

Our security definition is presented by the following game played by an adversary $\mathcal{A}$ against a challenger. The adversary takes as input the private key of a user Bob. The adversary is assumed to run in the random oracle model and is therefore given oracle access only to the hash functions H_1, H_2, H_3 and H_4. The adversary also has access to a certification oracle $\mathcal{O}$ that will supply it with authorisation credentials: given (i, s) for authority i and string s the oracle will respond with $C_i(s)$. Since the adversary is given access to the private key of the recipient we have no need to provide a decryption oracle $\mathcal{D}$ in this game. The adversary is assumed to be parameterised by a value $m \in \mathbb{N}$.

Stage 1: The adversary can adaptively query $\mathcal{O}$ on (i, s) pairs of its choosing. At the end of Stage 1 the adversary outputs two equal length messages M_0 and M_1, a policy $\mathcal{P}^*$ on m terms and a state S. We insist that for all qualifying sets in $\mathcal{P}^*$ there is at least one pair (i, s) for which the adversary has not queried $\mathcal{O}$ (otherwise its task is trivial).

Challenge: The simulator chooses a bit b. The the simulator, using the encryption algorithm above, encrypts M_b under Bob's public key and the policy $\mathcal{P}^*$. The resulting ciphertext C^* and the state S are passed back to the second stage of the adversary.

Stage 2: The adversary continues to query $\mathcal{O}$ on (i, s) pairs of its choosing, subject to the restriction that it cannot obtain from $\mathcal{O}$ credentials making up a qualifying set in $\mathcal{P}^*$.

Guess: At the end of Stage 2 the adversary outputs a bit b': its guess at b. The adversary is said to win the game if $b = b'$. We define the advantage of the adversary as

$$\mathrm{Adv}_{\mathrm{Rec}}(\mathcal{A}) = 2 \cdot \left| \Pr[b' = b] - \frac{1}{2} \right| = \left| \Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0] \right| .$$

Theorem 2. *Suppose that there is an adversary $\mathcal{A}$ of our scheme that makes at most q_i calls to random oracle i, that it makes at most q_o calls to $\mathcal{O}_{\mathcal{A}}$, then we have algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ that are of similar efficiency and such that*

$$\mathrm{Adv}_{\mathrm{Rec}}(\mathcal{A}) \leq 2 \cdot \mathrm{Adv}_{\mathrm{Enc}}(\mathcal{B}_3) + 2 \cdot \mathrm{Adv}_{\mathbb{S}_m}(\mathcal{B}_2) + 2 \cdot n \cdot q_1 \cdot q_2 \cdot \mathrm{Adv}_{\mathrm{BDHP}}(\mathcal{B}_1) + \frac{q_3 + q_4}{2^{l_s - 2}}.$$

Proof. The proof can be found in Appendix C.

6 Acknowledgements

We would like to thank Alex Dent for a carefull reading of an early version of this paper and his helpful comments and suggestions.

References

1. S.S. Al-Riyami. *Cryptographic schemes based on elliptic curve pairings*. Ph.D. Thesis, Royal Holloway, University of London, 2004.
2. S.S. Al-Riyami and K.G. Paterson. Certificateless public key cryptography. In *Advances in Cryptology – ASIACRYPT 2003*, Springer-Verlag LNCS 2894, 452–473, 2003.
3. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Advances in Cryptology – CRYPTO '88*, Springer-Verlag LNCS 403, 27–35, 1990.
4. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 213–229, 2001.
5. D. Boneh, B. Lynn and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – ASIACRYPT 2001*, Springer-Verlag LNCS 2248, 514–532, 2001.
6. R.W Bradshaw, J.E Holt and K.E. Seamons. Concealing complex policies with hidden credentials. In *11th ACM Conference on Computer and Communications Security*, 2004.
7. L. Chen, K. Harrison, D. Soldera and N.P. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *Infrastructure Security : InfraSec 2002*, Springer-Verlag LNCS 2437, 260–275, 2002

8. I. Duursma and H.-S. Lee. Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In *Advances in Cryptology – ASIACRYPT 2003*, Springer-Verlag LNCS 2894, 111–123, 2003.
9. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology – CRYPTO '99*, Springer-Verlag LNCS 1666, 537–554, 1999.
10. J.E Holt, R.W. Bradshaw, K.E. Seamons and H. Orman. Hidden credentials. In *2nd ACM Workshop on Privacy in the Electronic Society*, 1–8, 2003.
11. N. Li, W. Du and D. Boneh. Oblivious signature-based envelope. In *22nd ACM Symposium on Principles of Distributed Computing (PODC)*, 182–189, 2003.
12. K.G. Paterson. Cryptography from pairings: a snapshot of current research. *Information Security Technical Report*, **7**, 41–54, 2002.
13. C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology - CRYPTO '91*, Springer-Verlag LNCS 576, 433–444, 1992.
14. A. Shamir. How to share a secret. *Communications of the ACM*, **22**, 612–613, 1979.
15. V. Shoup. OAEP Reconsidered, In *Advances in Cryptology - CRYPTO 2001*, Springer-Verlag LNCS 2139, 239–259, 2001.
16. N.P. Smart. Access control using pairing based cryptography. In *Topics in Cryptology – CT-RSA 2003*, Springer-Verlag LNCS 2612, 111-121, 2003.

A Miscellany

Here we collect some comments and background which are not required to understand the scheme or its security definition, but which maybe required to understand either the security definition, the security proof, or how one should go about implementing our scheme efficiently.

A.1 The Decryption Oracle

The decryption oracle $\mathcal{D}$ above in both security definitions is assumed to only decrypt ciphertexts which are valid, irrespective of which authorising set of credentials they are decrypted with. This avoids a trivial malleability property, which we now explain. Suppose the challenge ciphertext C^* has the policy $\mathcal{P}^*$ with two terms given by

$$a \vee b.$$

Hence, the challenge ciphertext is given by

$$C^* = (U^*, c_1^*, c_2^*, A^*, \mathcal{P}^*, E^*).$$

The adversary could replace the value of c_2^* with a random bit string of length l, forming a ciphertext C. This ciphertext C is clearly different from C^* and when decrypted, using the credential corresponding to the term a in the policy $\mathcal{P}^*$, will result in the same message as that encrypted by C^*. However, when decrypted with the credential corresponding to the b term in the policy $\mathcal{P}^*$ the ciphertext is determined to be invalid. In our security model we assume that the decryption oracle will always return that C is invalid in such a situation.

Clearly the security model could be extended to allow the adversary to query a decryption oracle and specify which qualifying subset of credentials from the policy he would like to be used for the decryption. We feel this added security is perhaps too strong, and would require a far more complicated scheme than that explored in this paper. We therefore leave consideration of this case to future work.

A.2 The Security Model

We now discuss two issues which are raised by our wish to have escrow free workflow. It would appear that an escrow free workflow scheme must make use of a user private key, and so would appear to require the existance of user public keys to which one encrypts messages.

However, this escrow-free requirement restricts some of the applications of cryptographic workflow. One example is the broadcasting of data secured under a timelock, as considered in [7]. We note that, our system can be modified and then used as an escrowed system, and thereby enabling such examples as the broadcast timelock, by removing the computation of g and $H_2(g)$ (resp. g' and $H_2(g')$) in encryption (resp. decryption).

A security model for such a scheme can be obtained by keeping the same notion of recipient security, but alterring the notion of external security to a model in which the adversary does not know the private keys of the authorization authorities. Our security proofs then go over to this new situation with only minor changes. We note that the resulting escrowed scheme is very similar to the hybrid secret-sharing/OSBE system mentioned in [11]. However, our escrowed scheme has a rigourous security model and proof unlike the hybrid scheme of [11].

Our second issue related to the use of public keys is that the public keys in our scheme need to contain data corresponding to the authorization authorities which will be used to encrypt data. This means the encryptor is unable to choose the authorization authorities completely at will. Hence, it would be of some interest to construct a system which has a simpler public key structure. Indeed it would be interesting to devise an escrow-free workflow scheme which could be used with any existing deployed public keys. Such a generic construction is the topic of ongoing research.

A.3 Efficient Implementation

Encryption : We need to compute

$$g_j = \hat{t}([r]R_i, Q_{s_j}).$$

Note, if $\hat{t}(R_i, Q_{s_j})$ is already computed then it is more efficient to compute $\hat{t}(R_i, Q_{s_j})^r$. Also note that computations in $\mathbb{G}_2$ are usually more expensive than those in $\mathbb{G}_1$ for various groups, hence one should not compute g_j via

$$g_j = \hat{t}(R_i, [r]Q_{s_j}).$$

Decryption : We need to compute multiple

$$\hat{t}(U, C_i(s_j))$$

for all credentials $C_i(s_j)$ owned by the decryptor. Note, since U is a fixed first coordinate of all the Tate pairing computations, we can perform the Tate pairings in parallel to obtain increased efficiency improvements. However, this benefit is not so pronounced for the Duursma–Lee algorithm [8] in characteristic three, but is more noticeable for "standard" Miller-like algorithms for the Tate pairing in other characteristics.

Combining : Just as in [7] and [16] we can achieve a reduced number of pairings by adding together credentials which share a term in common and which are to form a conjunction in the policy, as the following illustrates:

Suppose our policy is in disjunctive normal form and that one of the inner terms (i.e. a conjunction) requires the decryptor to posses two credentials $C_i(s_1)$ and $C_i(s_2)$, i.e. credentials from the same authorisation authority but on different strings. The encryptor can in such a situation reduce the number of pairing computations by combining the two credentials together into a "virtual" credential by computing

$$g_j = \hat{t}([r]R_i, Q_{s_1} + Q_{s_2}).$$

The decryptor can achieve the same saving by computing

$$g_j = \hat{t}(U, C_i(s_1) + C_i(s_2)).$$

A similar saving can be made when the same string is signed by different authorities within a conjunctive term. For example the decryptor holds the credentials $C_i(s)$ and $C_{i'}(s)$. Encryption is then simplified by using a "virtual" credential by computing

$$g_j = \hat{t}([r](R_i + R_{i'}), Q_s).$$

The decryptor can again achieve the same efficiency improvement by computing

$$g_j = \hat{t}(U, C_i(s) + C_{i'}(s)).$$

A.4 One-More Oracle BDH Problem

In one of our security proofs, for the Recipient Security case, we will make use of a modified form of the BDH problem. This new definition implies that no matter how many solutions to the BDH problem one determines there are still problems which one cannot solve.

We define this notion via the following game: An adversary against what we shall call the *One-More Oracle BDH Problem* is given as input the following data

$$\{P, P', \{[x_i]P\}_{i=1}^n, \{[y_j]P'\}_{j=1}^m, [z]P\},$$

where $x_i, y_j, z \in \mathbb{F}_q^*$. The adversary is also given access to a Diffie–Hellman oracle $\mathcal{O}_{DH}$ which input of $[x_i]P$ and $[y_j]P'$, where $[x_i]P$ and $[y_j]P'$ are part of the input data, will output $[x_i y_j]P'$. At the end of the game the adversary should output a value of

$$\hat{t}(P, P')^{z x_i y_j},$$

subject to the constraint that $[x_i]P$ and $[y_j]P'$ form part of the input data and the query $\mathcal{O}_{DH}([x_i]P, [y_j]P')$ has not been made. To denote the dependence on n and m we denote the above problem by $\mathrm{OMBDH}(n, m)$. The advantage of such an adversary is defined in the standard way.

We have the following lemma relating security for the standard BDH problem and the One-More Oracle BDH problem.

Lemma 1. *Suppose that there is an adversary $\mathcal{A}$ against $\mathrm{OMBDH}(n, m)$ with advantage $\mathrm{Adv}_{\mathrm{OMBDH}}(\mathcal{A})$ making q_o calls to its $\mathcal{O}_{DH}$ oracle, we can construct an adversary $\mathcal{A}'$ against the standard BDH problem, just as efficient as $\mathcal{A}$, with advantage $\mathrm{Adv}_{\mathrm{BDHP}}(\mathcal{A}')$ such that*

$$\mathrm{Adv}_{\mathrm{BDHP}}(\mathcal{A}') \geq \frac{1}{nm} \mathrm{Adv}_{\mathrm{OMBDH}}(\mathcal{A}). \qquad (1)$$

Proof. Suppose that $\mathcal{A}$ is such an adversary against $\mathrm{OMBDH}(n, m)$ exists. We construct $\mathcal{A}'$ as follows: The algorithm $\mathcal{A}'$ takes as input $\{P, P, [x]P, [y]P', [z]P\}$ we then select $i_0 \in \{1, \dots, n\}$ and $j_0 \in \{1, \dots, m\}$ at random and define x_i and y_j for $i = 1, \dots, n$ and $j = 1, \dots, m$ to be random elements of $\mathbb{F}_q^*$. We set $P_i = [x_i]P$, except when $i = i_0$ in which case we set $P_i = [x]P$, we also set $P_j' = [y_j]P'$ except when $j = j_0$ in which case we set $P_j' = [y]P$.

We run $\mathcal{A}$ on input $\{P, P', \{P_i\}_{i=1}^n, \{P_j'\}_{j=1}^m, [z]P\}$. We use the known values x_i and y_i to answer the oracle queries $\mathcal{O}_{DH}(P_i, P_j')$ except when $(i, j) = (i_0, j_0)$ in which case we abort $\mathcal{A}$. When $\mathcal{A}$ outputs a value, we output the same value.

In the worst case we can assume that $\mathcal{A}$ makes $nm - 1$ such queries, and since i_0 and j_0 are out of the view of the adversary $\mathcal{A}$ we have

$$\mathrm{Adv}_{\mathrm{BDHP}}(\mathcal{A}') \geq \frac{1}{nm} \mathrm{Adv}_{\mathrm{OMBDH}}(\mathcal{A}).$$

A.5 A Preliminary Lemma

In our security proofs, in the following appendices, we will make frequent of the following useful lemma from [15].

Lemma 2. *Let E, F and G be events defined on a probability space such that $\Pr[E \wedge \neg G] = \Pr[F \wedge \neg G]$. We have*

$$|\Pr[E] - \Pr[F]| \leq \Pr[G].$$

B Proof of Theorem 1

Our proof strategy is as follows. We define a sequence $\mathbf{G}_0, \ldots, \mathbf{G}_3$ of modified attack games. The only difference between games is how the simulated environment responds to $\mathcal{A}$'s oracle queries. For any $0 \leq i \leq 4$, we let S_i be the probability that $b' = b$ in $\mathbf{G}_i$. In certain games the simulator will use an instance of the BDH problem

$$\{P, P', [x]P, [y]P', [z]P\}. \tag{2}$$

Game $\mathbf{G_0}$

This is the real attack game and so

$$\Pr[S_0] = \frac{1}{2}\mathrm{Adv}_{\mathrm{Ext}}(\mathcal{A}) + \frac{1}{2}. \tag{3}$$

Game $\mathbf{G_1}$

In this game we modify the environment so that $\mathcal{A}$ now interacts with a simulator $\mathcal{B}$. First of all $\mathcal{B}$ runs the various key generation algorithms, it then passes $\mathcal{A}$ the public key (X, Q), plus the set of public/private key pairs of the authorisation authorities $\{R_i, \mathfrak{a}_i\}_{i=1}^n$.

We define how our simulators $\mathcal{B}$ responds to the various oracle calls of $\mathcal{A}$ below. Note that algorithm $\mathcal{B}$ uses $[y]P'$ from the BDH instance (2) in responding to the H_1 queries.

H_1 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_1$ of triples $(s_k, H_{1,k}, b_k) \in \{0,1\}^* \times \mathbb{G}_2 \times \mathbb{F}_q^*$ which is initially empty. If a query s is made to H_1, then $\mathcal{B}$ checks whether s occurs as a first component s_k in the list, if so the output of H_1 is the corresponding value $H_{1,k}$. Otherwise b_s is chosen uniformly at random from $\mathbb{F}_q^*$ and $Q_s = [b_s]([y]P')$ is computed. The triple (s, Q_s, b_s) is then added to $\mathcal{H}_1$ and Q_s is returned to $\mathcal{A}$.

H_2 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_2$ of pairs $(g_k, H_{2,k}) \in \mathbb{G}_T \times \{0,1\}^l$. If a query g is made to H_2, then $\mathcal{B}$ checks whether g occurs as the first component g_k in the list, if so the output of H_2 is the corresponding value $H_{2,k}$. Otherwise, H is chosen uniformly at random from $\{0,1\}^l$, (g, H) is placed on the list and H is returned to $\mathcal{A}$.

H_3 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_3$ of pairs $(s_k, H_{3,k}) \in \{0,1\}^* \times \mathbb{F}_q^*$. If a query $s = \mathfrak{b}\|\mathcal{P}\|M$ is made to H_3, then $\mathcal{B}$ checks whether s occurs as the first component s_k in the list, if so the output of H_3 is the corresponding value $H_{3,k}$. Otherwise, H is chosen uniformly at random from $\mathbb{F}_q^*$, (s, H) is placed on the list and H is returned to $\mathcal{A}$.

H_4 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_4$ of pairs $(s_k, H_{4,k}) \in \{0,1\}^{l_s} \times \{0,1\}^{l_e}$. If a query s is made to H_4, then $\mathcal{B}$ checks whether s occurs as the first component s_k in the list, if so the output of H_4 is the corresponding value $H_{4,k}$. Otherwise, H is chosen uniformly at random from $\{0,1\}^{l_e}$, (s, H) is placed on the list and H is returned to $\mathcal{A}$.

$\mathcal{D}$ **Oracle Queries:** Algorithm $\mathcal{B}$ can answer these using the genuine decryption algorithm since at this point it knows all the necessary private keys. When doing so it replaces hash function calls with calls to the oracles described above.

When $\mathcal{A}$ outputs $\mathcal{P}^*$, M_0 and M_1 on which it wishes to be challenged, $\mathcal{B}$ chooses a bit b and encrypts M_b under $\mathcal{P}^*$, replacing hash function calls with calls to the oracles above. Let $(U^*, \{c_j^*\}_{j=1}^m, A^*, \mathcal{P}^*, E^*)$ be the resulting ciphertext; $\mathcal{B}$ returns this to $\mathcal{A}$.

These simulators are clearly consistent and so, in the random oracle model we have

$$\Pr[S_1] = \Pr[S_0]. \tag{4}$$

Game $\mathbf{G_2}$

In this game we are going to embed more of the BDH problem instance (2) into our simulation.

To begin with we define the public key given to the adversary as follows.

- Select $\mathfrak{a}_i \in \mathbb{F}_q^*$ at random and set $R_i = [\mathfrak{a}_i]P$, for $1 \leq i \leq n$.
- Set $X = [x]P$.

Since we no longer know the private key of the user (x in our updated simulation), it is now necessary to modify our decryption oracle. In this game we use the following oracle to decrypt a ciphertext $(U, \{c_j\}_{j=1}^{m'}, A, \mathcal{P}, E)$, where $\mathcal{P}$ is a policy on m' terms $\{(i_j, s_j)\}_{j=1}^{m'}$

1. Search the $\mathcal{H}_3$ until a pair $(s_{3_k}, H_{3,k})$ is found such that $[H_{3,k}]P = U$ or until the end of the list is reached. If no such entry exists, return $\perp$, else let $r = H_{3,k}$.
2. For $j = 1, \ldots, m'$ let $Q_j = H_1(s_j)$ where the values s_j are from policy $\mathcal{P}$ and H_1 represents the simulator above.
3. Set $g = \hat{t}([r]X, Q)$.
4. For $j = 1, \ldots, m'$ let $g_j = \hat{t}([r]R_i, Q_j)$.
5. For $j = 1, \ldots, m'$ let $\mathfrak{b}_j = c_j \oplus H_2(g_j) \oplus H_2(g)$ where H_2 is the simulator for H_2.
6. Let $\mathbb{K}' = \{\mathfrak{b}_j\}_{j=1}^{m'}$, compute $\mathfrak{b} = \mathbb{S}_m^{-1}(\mathbb{K}, A)$, if $\mathfrak{b} = \perp$ return $\perp$.
7. Compute $M = \mathrm{Enc}_{H_4(\mathfrak{b})}^{-1}(E)$.
8. If the pair $(\mathfrak{b}\|\mathcal{P}\|M, r)$ does not appear in the $\mathcal{H}_3$ list return $\perp$, else return M.

It is easy to verify that $\mathbf{G}_2$ proceeds exactly as $\mathbf{G}_1$ until one of two events occurs: (1) there is a collision in the H_3 oracle , or (2) the decryption simulator rejects a valid ciphertext. We denote these events Coll_2 and Reject_2 respectively. We have

$$\Pr[\mathsf{Coll}_2] \leq q_3(q_3 - 1)/2(q - 1).$$

Also, the only valid ciphertexts that are rejected are those that verify without the requisite H_3 query being made. Therefore

$$\Pr[\mathsf{Reject}_2] \leq q_d/(q - 1).$$

Putting this together using Lemma 2 we have

$$|\Pr[S_2] - \Pr[S_1]| \leq \frac{q_3(q_3 - 1)}{2(q - 1)} + \frac{q_d}{q - 1}. \tag{5}$$

Game $\mathbf{G}_3$

This game proceeds in very much the same way as game $\mathbf{G}_2$, the difference is in how the challenge ciphertext is created. At the beginning of this game $\mathcal{B}$ chooses $\mathfrak{b}^*$ at random from $\{0,1\}^{l_s}$ as it would do when creating the challenge ciphertext.

Again using the BDH instance (2), the challenge ciphertext for $(\mathcal{P}^*, M_0, M_1)$ output by $\mathcal{A}$ is now created as follows.

1. Choose b at random from $\{0, 1\}$.
2. Compute $(\mathbb{K}^*, A^*) = \mathbb{S}_m(\mathcal{P}^*, \mathfrak{b}^*)$.
3. Choose κ^* at random from $\{0,1\}^{l_e}$ (where l_e is the output length of H_4)
4. Set $E^* = \mathrm{Enc}_{\kappa^*}(M_b)$.
5. Set $U^* = [z]P$.
6. For $j = 1, \ldots, m$, choose c_j^* at random from $\{0,1\}^l$.
7. Return $(U^*, \{c_j^*\}_{j=1}^m, A^*, \mathcal{P}^*, E^*)$.

For the $\{(i_j, s_j)\}_{j=1}^m$ contained in policy $\mathcal{P}^*$, let

$$Q_j = H_1(s_j) = [b_{s_j} y]P'$$

and $Q = [b'y]P'$ for the integers b' and b_{s_j} generated by the simulation for H_1. Also, let $\mathbb{K}^* = \{\mathfrak{b}'_j\}_{j=1}^m$.

For the challenge ciphertext to be consistent with the rest of the simulation we should have, for $j = 1, \ldots, m$

$$\begin{aligned} \mathfrak{b}'_j &= c_j^* \oplus H_2\left(\hat{t}([z]R_{i_j}, Q_j)\right) \oplus H_2\left(\hat{t}([z]X, Q\right) \\ &= c_j^* \oplus H_2\left(\hat{t}(P, P')^{yz\mathfrak{a}_{i_j} b_{s_j}}\right) \oplus H_2\left(\hat{t}(P, P')^{xyzb'}\right). \end{aligned} \tag{6}$$

Let us denote the event that the query

$$H_2\left(\hat{t}(P, P')^{xyzb'}\right) \tag{7}$$

is made by AskBDH_3. This event will go undetected by our simulator and so the response of the H_2 oracle will be incorrect if it occurs. From (6) above we see that it should be equal to

$$\mathfrak{b}'_j \oplus c^*_j \oplus H_2\left(\hat{t}(P,P')^{yz\mathfrak{a}_{i_j}b_{s_j}}\right)$$

for all values of $j \in \{1,\ldots,m\}$.

The other consistency constraints imposed by the creation of the challenge ciphertext is that we should have

$$z = H_3(\mathfrak{b}^*||\mathcal{P}^*||M_b) \text{ and } \kappa^* = H_4(\mathfrak{b}^*)$$

However, as it is constructed $\mathcal{B}$ will not respond in this way.

Let us denote the event that $\mathcal{A}$ makes a query $\mathfrak{b}^*||str$ to the H_3 oracle for some string str, or it makes the query $\mathfrak{b}^*$ to H_4, by AskBS_3. Let AskBS_3^1 be the event that AskBS_3 occurs before the challenge ciphertext is given to $\mathcal{A}$ and let AskBS_3^2 be the event that AskBS_3 occurs after the challenge ciphertext is given to $\mathcal{A}$.

Examining our simulation, we see $\mathbf{G}_3$ proceeds exactly as $\mathbf{G}_2$ until AskBDH_3 or AskBS_3 occurs. Therefore, by Lemma 2 we have

$$\begin{aligned}
&|\Pr[S_3] - \Pr[S_2]| \\
&\qquad \le \Pr[\mathsf{AskBDH}_3 \vee \mathsf{AskBS}_3] \\
&\qquad = \Pr[\mathsf{AskBDH}_3] + \Pr[\mathsf{AskBS}_3 \wedge \neg\mathsf{AskBDH}_3] \\
&\qquad \le \Pr[\mathsf{AskBDH}_3] + \Pr[\mathsf{AskBS}_3|\neg\mathsf{AskBDH}_3] \\
&\qquad = \Pr[\mathsf{AskBDH}_3] + \Pr[\mathsf{AskBS}_3^1|\neg\mathsf{AskBDH}_3] + \Pr[\mathsf{AskBS}_3^2|\neg\mathsf{AskBDH}_3] \\
&\qquad \le (q_3+q_4)/2^{l_s} + \Pr[\mathsf{AskBDH}_3] + \Pr[\mathsf{AskBS}_3^2|\neg\mathsf{AskBDH}_3]. \qquad (8)
\end{aligned}$$

The last inequality follows from the fact that, before the challenge ciphertext is given to $\mathcal{A}$, it has no information about $\mathfrak{b}^*$.

Now, let $\mathcal{B}_1$ be the algorithm that begins by running $\mathcal{A}$ exactly as it is run in $\mathbf{G}_3$. Once the simulation terminates it retrieves (X, Q, b') from the $\mathcal{H}_1$ list. Next it chooses a random input g^* from the $\mathcal{H}_2$ list and computes

$$(g^*)^{1/b' \bmod q}.$$

In the event that AskBDH_3 (as defined in (7) above) has occurred, this is the solution to the BDH instance $\{P, P', [x]P, [y]P', [z]P\}$ with probability $1/q_2$. We conclude that

$$\Pr[\mathsf{AskBDH}_3] \le q_2 \cdot \mathrm{Adv}_{BDH}(\mathcal{B}_1). \qquad (9)$$

To bound $\Pr[\mathsf{AskBS}_3^2|\neg\mathsf{AskBDH}_3]$ we construct an adversary $\mathcal{B}_2$ of the secret sharing scheme $\mathbb{S}_m$. We do this describing a modified version of the game $\mathbf{G}_3$.

Game $\mathbf{G}_3'$

In this game we describe how $\mathcal{A}$ can be used to construct an adversary $\mathcal{B}_2$ of $\mathbb{S}_m$. At the beginning of this game $\mathcal{B}_2$ chooses $\mathfrak{b}_0^*$ and $\mathfrak{b}_1^*$ at random from $\{0,1\}^l$. It runs $\mathcal{A}$ in a very similar manner to how $\mathcal{A}$ is run in $\mathbf{G}_3$ except that, once it has given $\mathcal{A}$ the challenge ciphertext, it modifies the H_3 and H_4 oracles as follows.

H_3 **Oracle Queries:** If queried with s such that $s = \mathfrak{b}_0^* \| str_0$ for some str_0, return 0 and terminate the simulation. If queried with s such that $s = \mathfrak{b}_1^* \| str_1$ for some str_1, return 1 and terminate the simulation. Otherwise, respond as in $\mathbf{G}_3$.

H_4 **Oracle Queries:** If queried with $\mathfrak{b}_0^*$, return 0 and terminate the simulation. If queried with $\mathfrak{b}_1^*$, return 1 and terminate the simulation. Otherwise, respond as in $\mathbf{G}_3$.

Using the BDHP problem instance $\{P, P', [x]P, [y]P', [z]P\}$, the challenge ciphertext for $(\mathcal{P}^*, M_0, M_1)$ output by $\mathcal{A}$ is now generated as follows.

1. Choose b at random from $\{0,1\}$.
2. Call $\mathcal{B}_2$'s challenge oracle with $(\mathcal{P}^*, \mathfrak{b}_0^*, \mathfrak{b}_1^*)$ to obtain A^*.
3. Set $E^* = \mathrm{Enc}_{\kappa^*}(M_b)$.
4. Set $U^* = [z]P$.
5. For $j = 1, \ldots, m$, choose c_j^* at random from $\{0,1\}^l$.
6. Return $(U^*, \{c_j^*\}_{j=1}^m, A^*, \mathcal{P}^*, E^*)$.

Let d be the hidden bit of $\mathcal{B}_2$'s challenge oracle and let d' be the output of $\mathcal{B}_2$. By definition we have

$$\mathrm{Adv}_{\mathbb{S}_m}(\mathcal{B}_2) = |\Pr[d' = 1|d = 1] - \Pr[d' = 1|d = 0]|.$$

Our construction gives us

$$\Pr[d' = 1|d = 1] = \Pr[\mathsf{AskBS}_3^2|\neg\mathsf{AskBDH}_3].$$

Also, since $\mathfrak{b}_{\bar{b}}$ is hidden from $\mathcal{A}$'s view we have

$$\Pr[d' = 1|d = 0] \le (q_3 + q_4)/2^{l_s}$$

It follows that

$$\Pr[\mathsf{AskBS}_3^2|\neg\mathsf{AskBDH}_3] \le \mathrm{Adv}_{\mathbb{S}_m}(\mathcal{B}_2) + (q_3 + q_4)/2^{l_s}. \tag{10}$$

To complete the proof we show that there is an algorithm $\mathcal{B}_3$ to break the FG security of Enc such that

$$\frac{\mathrm{Adv}_{\mathrm{Enc}}(\mathcal{B}_3)}{2} + \frac{1}{2} = \Pr[S_3]. \tag{11}$$

This algorithm runs $\mathcal{A}$ as it would be run in game $\mathbf{G}_3$ except that the challenge ciphertext is now generated as follows.

1. Choose $\mathfrak{b}^*$ at random from $\{0,1\}^{l_s}$.
2. Compute $(\mathbb{K}^*, A^*) = \mathbb{S}_m(\mathcal{P}^*, \mathfrak{b}^*)$.
3. Call the challenge oracle the for FG attack game with (M_0, M_1) and receive ciphertext E^* in response.
4. Set $U^* = [z]P$.
5. For $j = 1, \ldots, m$, choose c_j^* at random from $\{0,1\}^l$.
6. Return $(U^*, \{c_j^*\}_{j=1}^m, A^*, \mathcal{P}^*, E^*)$.

Let d be the hidden bit of $\mathcal{B}_3$'s challenge oracle. By construction we have the following.

$$\Pr[d' = 1 | d = 1] = \Pr[S_3]$$

By definition this gives us

$$\frac{\mathrm{Adv}_{\mathrm{Enc}}(\mathcal{B}_3)}{2} + \frac{1}{2} = \Pr[S_3]$$

as required.

The result now follows from (3), (4), (5), (8), (9), (10) and (11).

C Proof of Theorem 2

The philosophy behind this proof is similar to that of the previous proof, but now in certain games the simulator will use an instance of the OMBDH(n, q_1) problem

$$\{P, P', \{[x_i]P\}_{i=1}^n, \{[y_j]P'\}_{j=1}^{q_1}, [z]P\}.$$

Game $\mathbf{G_0}$

This is the real attack game and so

$$\Pr[S_0] = \frac{1}{2}\mathrm{Adv}_{\mathrm{Rec}}(\mathcal{A}) + \frac{1}{2}. \tag{12}$$

Game $\mathbf{G_1}$

In this game we modify the environment so that $\mathcal{A}$ now interacts with a simulator $\mathcal{B}$. First of all $\mathcal{B}$ runs the various key generation algorithms, it then passes $\mathcal{A}$ the public key (X, Q) and the corresponding private key $\mathfrak{u}$, plus the public keys of the authorisation authorities $\{R_i\}_{i=1}^n$.

We define how our simulators $\mathcal{B}$ responds to the various oracle calls of $\mathcal{A}$ below.

H_1 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_1$ of triples $(s_k, H_{1,k}, b_k) \in \{0,1\}^* \times \mathbb{G}_2 \times \mathbb{F}_q^*$ which is initially empty. If a query s is made to H_1, then $\mathcal{B}$

checks whether s occurs as a first component s_k in the list, if so the output of H_1 is the corresponding value $H_{1,k}$. Otherwise b_s is chosen uniformly at random from $\mathbb{F}_q^*$ and $Q_s = [b_s]P'$ is computed. The triple (s, Q_s, b_s) is then added to $\mathcal{H}_1$ and Q_s is returned to $\mathcal{A}$.

H_2 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_2$ of pairs $(g_k, H_{2,k}) \in \mathbb{G}_T \times \{0,1\}^l$. If a query g is made to H_2, then $\mathcal{B}$ checks whether g occurs as the first component g_k in the list, if so the output of H_2 is the value $H_{2,k}$. Otherwise, H is chosen uniformly at random from $\{0,1\}^l$, (g, H) is placed on the list and H is returned to $\mathcal{A}$.

H_3 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_3$ of pairs $(s_k, H_{3,k}) \in \{0,1\}^* \times \mathbb{F}_q^*$. If a query $s = \mathfrak{b}\|\mathcal{P}\|M$ is made to H_3, then $\mathcal{B}$ checks whether s occurs as the first component s_k in the list, if so the output of H_3 is the value $H_{3,k}$. Otherwise, H is chosen uniformly at random from $\mathbb{F}_q^*$, (s, H) is placed on the list and H is returned to $\mathcal{A}$.

H_4 **Oracle Queries:** Algorithm $\mathcal{B}$ maintains a list $\mathcal{H}_4$ of pairs $(s_k, H_{4,k}) \in \{0,1\}^{l_s} \times \{0,1\}^{l_e}$. If a query s is made to H_4, then $\mathcal{B}$ checks whether s occurs as the first component s_k in the list, if so the output of H_4 is the value $H_{4,k}$. Otherwise, H is chosen uniformly at random from $\{0,1\}^{l_e}$, (s, H) is placed on the list and H is returned to $\mathcal{A}$.

$\mathcal{O}$ **Oracle Queries:** Suppose $\mathcal{B}$ is responding to the query (i, s) where i denotes the ith authorization authority. We can assume that before the adversary makes such a query it has already made the query $H_1(s)$, otherwise $\mathcal{B}$ can make this query itself.

The first thing that $\mathcal{B}$ does is to recover the entry (s, Q_s, b_s) from $\mathcal{H}_1$. It then computes $C_i(s) = [\mathfrak{a}_i b_s]P'$. It records $(i, s, C_i(s))$ in a list $\mathcal{C}$ and returns $C_i(s)$ to $\mathcal{A}$.

When $\mathcal{A}$ outputs $\mathcal{P}^*$, M_0 and M_1 on which it wishes to be challenged, $\mathcal{B}$ chooses a bit b and encrypts M_b under $\mathcal{P}^*$, replacing hash function calls with calls to the oracles above. Let $(U^*, \{c_j^*\}_{j=1}^m, A^*, \mathcal{P}^*, E^*)$ be the resulting ciphertext; $\mathcal{B}$ returns this to $\mathcal{A}$.

These simulators are clearly consistent and so, in the random oracle model we have

$$\Pr[S_1] = \Pr[S_0]. \tag{13}$$

Game $\mathbf{G_2}$

The only changes between this game and $\mathbf{G}_1$ are (1) how the challenge ciphertext is generated and (2) how $\mathcal{B}$ deals with the $\mathcal{O}$ oracle.

Suppose that $\mathcal{A}$ outputs $(\mathcal{P}^*, M_0, M_1)$, using $[z]P$ for z chosen at random from $\mathbb{F}_p^*$ and unknown to $\mathcal{B}$, the challenge ciphertext is generated as follows.

1. Choose b at random from $\{0,1\}$.
2. Chooses $\mathfrak{b}^*$ at random from $\{0,1\}^{l_s}$.
3. Compute $(\mathbb{K}^*, A^*) = \mathbb{S}_m(\mathcal{P}^*, \mathfrak{b}^*)$, and store $\mathbb{K}^* = \{\mathfrak{b}'_1, \ldots, \mathfrak{b}'_n\}$.
4. For $j = 1, \ldots, m$, choose c_j^* at random from $\{0,1\}^l$.
5. Set $U^* = [z]P$.
6. Set $g = \hat{t}([\mathfrak{u}]U^*, Q)$.
7. Choose $\mathfrak{b}'$ at random from $\{0,1\}^l$ and add $(g, \mathfrak{b}')$ to $\mathcal{H}_2$.
8. For every $(i, s) \in \mathcal{P}^* \cap \mathcal{C}$ (where $\mathcal{C}$ is the list maintained by the $\mathcal{O}$ oracle), compute $g_i = \hat{t}(U^*, C_i(s))$ and add $(g_i, c_i^* \oplus \mathfrak{b}'_i \oplus \mathfrak{b}')$ to $\mathcal{H}_2$.
9. Choose κ^* at random from $\{0,1\}^{l_e}$.
10. Set $E^* = \mathrm{Enc}_{\kappa^*}(M_b)$.
11. Return $(U^*, \{c_j^*\}_{i=1}^m, A^*, \mathcal{P}^*, E^*)$.

$\mathcal{O}$ **Oracle Queries:** If the query (i, s) is being made before the challenge ciphertext is issued, respond exactly as in $\mathbf{G}_1$.

Otherwise, proceed exactly as in $\mathbf{G}_1$, but before responding to $\mathcal{A}$, check to see if $(i, s) \in \mathcal{P}^*$. If so, compute $g_i = \hat{t}(U^*, C_i(s))$ and add $(g_i, c_i^* \oplus \mathfrak{b}'_i \oplus \mathfrak{b}')$ to $\mathcal{H}_2$.

Let us denote by $\mathsf{AskH2}_2$ the event that H_2 is called for a value of g_j and subsequently in the generation of the challenge ciphertext or in responding to an $\mathcal{O}$ query we have $g_j = \hat{t}(U^*, C_i(s))$. Examining the simulations above we see that in the event $\mathsf{AskH2}_2$ we get we get an inconsistency in H_2.

The other constraints imposed by the creation of the challenge ciphertext is that we should have

$$z = H_3(\mathfrak{b}^* || \mathcal{P}^* || M_b) \text{ and } \kappa^* = H_4(\mathfrak{b}^*)$$

However, as it is constructed $\mathcal{B}$ will not respond in this way.

Let us denote the event that $\mathcal{A}$ makes a query $\mathfrak{b}^* || str$ to the H_2 oracle for some string str, or it makes the query $\mathfrak{b}^*$ to H_4, by AskBS_2. Let AskBS_2^1 be the event that AskBS_2 occurs before the challenge ciphertext is given to $\mathcal{A}$ and let AskBS_2^2 be the event that AskBS_2 occurs after the challenge ciphertext is given to $\mathcal{A}$.

Examining our simulation, we see $\mathbf{G}_2$ proceeds exactly as $\mathbf{G}_2$ until $\mathsf{AskH2}_2$ or AskBS_2 occurs. Therefore, by Lemma 2 we have

$$\begin{aligned}
&|\Pr[S_2] - \Pr[S_1]| \\
&\qquad \leq \Pr[\mathsf{AskH2}_2 \vee \mathsf{AskBS}_2] \\
&\qquad = \Pr[\mathsf{AskH2}_2] + \Pr[\mathsf{AskBS}_2 \wedge \neg\mathsf{AskH2}_2] \\
&\qquad \leq \Pr[\mathsf{AskH2}_2] + \Pr[\mathsf{AskBS}_2 | \neg\mathsf{AskH2}_2] \\
&\qquad = \Pr[\mathsf{AskH2}_2] + \Pr[\mathsf{AskBS}_2^1 | \neg\mathsf{AskH2}_2] + \Pr[\mathsf{AskBS}_2^2 | \neg\mathsf{AskH2}_2] \\
&\qquad \leq (q_2 + q_4)/2^{l_s} + \Pr[\mathsf{AskH2}_2] + \Pr[\mathsf{AskBS}_2^2 | \neg\mathsf{AskH2}_2]. \qquad (14)
\end{aligned}$$

The last inequality follows from the fact that, before the challenge ciphertext is given to $\mathcal{A}$, it has no information about $\mathfrak{b}^*$.

We now bound $\Pr[\mathsf{AskH2}_2]$ by constructing an algorithm $\mathcal{B}_1$ to solve an instance $\{P, P', \{[x_i]P\}_{i=1}^n, \{[y_j]P'\}_{j=1}^{q_1}, [z]P\}$ of the OMBDH problem. Algorithm $\mathcal{B}_1$ simulates $\mathcal{A}$ in a very similar manner to the way $\mathcal{B}$ simulates $\mathcal{A}$ in game $\mathbf{G}_2$. The differences are as follows.

– $\mathcal{B}_1$ replaces $R_i = [\mathfrak{u}_i]P$ with $R_i = [x_i]P$ for the authorisation authority public keys.
– $\mathcal{B}_1$ responds to the j-th query to H_1 with $[y_j]P'$ and does the appropriate bookkeeping.
– $\mathcal{B}_1$ responds to the $\mathcal{O}$ queries using its DH oracle for the OMBDH problem by computing $C_i(s) = \mathcal{O}_{DH}([x_i]P, [y_j]P')$ where $[y_j]P' = H_1(s)$.
– $\mathcal{B}_1$ chooses j^* at random from $\{1, \ldots, q_2\}$ and when it receives g_{j^*}, the j^*-th query to H_2, it stops the simulation and outputs g_{j^*}.

Now, $\mathcal{A}$'s view when simulated by $\mathcal{B}$ in game $\mathbf{G}_2$ and when simulated by $\mathcal{B}_1$ here is identical up to the point when $\mathcal{A}$ makes the j^*-th query to H_2. Moreover, in the event $\mathsf{AskH2}_2$, $\mathcal{B}_1$ succeeds in solving the OMBDH problem with probability $1/q_2$. We conclude that

$$\Pr[\mathsf{AskH2}_2] \leq q_2 \mathrm{Adv}_{\mathrm{OMBDH}}(\mathcal{B}_1). \tag{15}$$

To bound $\Pr[\mathsf{AskBS}_2^2 | \neg\mathsf{AskH2}_2]$ we construct an adversary $\mathcal{B}_2$ of the secret sharing scheme $\mathbb{S}_m$. We do this describing a modified version of the game $\mathbf{G}_2$.

Game $\mathbf{G}_2'$

In this game we describe how $\mathcal{A}$ can be used to construct an adversary $\mathcal{B}_2$ of $\mathbb{S}_m$. At the beginning of this game $\mathcal{B}_2$ chooses $\mathfrak{b}_0^*$ and $\mathfrak{b}_1^*$ at random from $\{0,1\}^l$. It runs $\mathcal{A}$ in a very similar manner to how $\mathcal{A}$ is run in $\mathbf{G}_2$ except that, once it has given $\mathcal{A}$ the challenge ciphertext, it modifies the H_3 and H_4 oracles as follows.

H_3 **Oracle Queries:** If queried with s such that $s = \mathfrak{b}_0^* \| str_0$ for some str_0, return 0 and terminate the simulation. If queried with s such that $s = \mathfrak{b}_1^* \| str_1$ for some str_1, return 1 and terminate the simulation. Otherwise, respond as in $\mathbf{G}_3$.

H_4 **Oracle Queries:** If queried with $\mathfrak{b}_0^*$, return 0 and terminate the simulation. If queried with $\mathfrak{b}_1^*$, return 1 and terminate the simulation. Otherwise, respond as in $\mathbf{G}_3$.

The challenge ciphertext for $(\mathcal{P}^*, M_0, M_1)$ output by $\mathcal{A}$ is now generated as follows by $\mathcal{B}_2$.

1. Choose b at random from $\{0,1\}$.
2. Call $\mathcal{B}_2$'s challenge oracle with $(\mathcal{P}^*, \mathfrak{b}_0^*, \mathfrak{b}_1^*)$ to obtain A^*.

3. Choose κ^* at random from $\{0,1\}^{l_e}$ (where l_e is the output length of H_4).
4. Set $E^* = \mathrm{Enc}_{\kappa^*}(M_b)$.
5. Set $U^* = [z]P$.
6. Set $g = \hat{t}([\mathfrak{u}]U^*, Q)$.
7. Choose $\mathfrak{b}'$ at random from $\{0,1\}^l$ and add $(g, \mathfrak{b}')$ to $\mathcal{H}_2$.
8. For every $(i,s) \in \mathcal{P}^* \cap \mathcal{C}$ (where $\mathcal{C}$ is the list maintained by the $\mathcal{O}$ oracle), call $\mathcal{B}_2$'s oracle to obtain $\mathfrak{b}'_i$, compute $g_i = \hat{t}(U^*, C_i(s))$ and add $(g_i, c_i^* \oplus \mathfrak{b}'_i \oplus \mathfrak{b}')$ to $\mathcal{H}_2$.
9. For $j = 1, \ldots, m$, choose c_j^* at random from $\{0,1\}^l$.
10. Return $(U^*, \{c_j^*\}_{i=1}^m, A^*, \mathcal{P}^*, E^*)$.

$\mathcal{O}$ **Oracle Queries:** If the query (i,s) is being made before the challenge ciphertext is issued, respond exactly as in $\mathbf{G}_2$.

Otherwise, proceed exactly as in $\mathbf{G}_2$, but before responding to $\mathcal{A}$, check to see if $(i,s) \in \mathcal{P}^*$. If so, call $\mathcal{B}_2$'s oracle to obtain $\mathfrak{b}'_i$, compute $g_i = \hat{t}(U^*, C_i(s))$ and add $(g_i, c_i^* \oplus \mathfrak{b}'_i \oplus \mathfrak{b}')$ to $\mathcal{H}_2$.

Let d be the hidden bit of $\mathcal{B}_2$'s challenge oracle and let d' be the output of $\mathcal{B}_2$. By definition we have

$$\mathrm{Adv}_{\mathbb{S}_m}(\mathcal{B}_2) = |\Pr[d' = 1 | d = 1] - \Pr[d' = 1 | d = 0]|$$

Our construction gives us

$$\Pr[d' = 1 | d = 1] = \Pr[\mathsf{AskBS}_2^2 | \neg \mathsf{AskH2}_2].$$

Also, since $\mathfrak{b}_{\bar{b}}$ is hidden from $\mathcal{A}$'s view we have

$$\Pr[d' = 1 | d = 0] \leq (q_3 + q_4)/2^{l_s}$$

It follows that

$$\Pr[\mathsf{AskBS}_2^2 | \neg \mathsf{AskH2}_2] \leq \mathrm{Adv}_{\mathbb{S}_m}(\mathcal{B}_2) + (q_3 + q_4)/2^{l_s}. \tag{16}$$

To complete the proof we bound $\Pr[S_2]$. To do this we construct an adversary $\mathcal{B}_3$ of Enc by simulating $\mathcal{A}$ as $\mathcal{B}$ does in $\mathbf{G}_2$. The only difference being how we create the challenge ciphertext. This is done as follows.

1. Chooses $\mathfrak{b}^*$ at random from $\{0,1\}^{l_s}$.
2. Compute $(\mathbb{K}^*, A^*) = \mathbb{S}_m(\mathcal{P}^*, \mathfrak{b}^*)$, and store $\mathbb{K}^* = \{\mathfrak{b}'_1, \ldots, \mathfrak{b}'_m\}$.
3. For $j = 1, \ldots, m$, choose c_j^* at random from $\{0,1\}^l$.
4. Set $U^* = [z]P$.
5. Set $g = \hat{t}([\mathfrak{u}]U^*, Q)$.
6. Choose $\mathfrak{b}'$ at random from $\{0,1\}^l$ and add $(g, \mathfrak{b}')$ to $\mathcal{H}_2$.
7. For every $(i,s) \in \mathcal{P}^* \cap \mathcal{C}$ (where $\mathcal{C}$ is the list maintained by the $\mathcal{O}$ oracle), compute $g_i = \hat{t}(U^*, C_i(s))$ and add $(g_i, c_i^* \oplus \mathfrak{b}'_i \oplus \mathfrak{b}')$ to $\mathcal{H}_2$.
8. Call $\mathcal{B}_3$'s challenge oracle with (M_0, M_1) to obtain E^*.
9. Return $(U^*, \{c_j^*\}_{i=1}^m, A^*, \mathcal{P}^*, E^*)$.

Let d be the hidden bit of $\mathcal{B}_3$'s challenge oracle. By construction we have the following.

$$\Pr[d' = 1 | d = 1] = \Pr[S_2]$$

By definition this gives us

$$\frac{\mathrm{Adv}_{\mathrm{Enc}}(\mathcal{B}_3)}{2} + \frac{1}{2} = \Pr[S_2]. \tag{17}$$

The result now follows from (12), (13), (14), (15), (16) and (17).