Institut EURECOM
Corporate Communications
2229, route des Crêtes BP 193
06904 Sophia Antipolis
(France)

Research Report[a] N^o 141 — RR-05-141

Provably Secure Policy-Based Cryptography

Walid Bagga and Refik Molva

May, 2005

[a] Eurecoms research is partially supported by its industrial partners: Bouygues Telecom, Fondation d'entreprise Groupe Cegetel, Fondation Hasler, France Telecom, Hitachi, ST Microelectronics, Swisscom, Texas Instruments, and Thales

Provably Secure Policy-Based Cryptography

Abstract. The concept of *policy-based cryptography* (PBC) is a promising paradigm for trust establishment and authorization in large-scale open environments. A policy-based encryption scheme (PBE) allows to encrypt a message according to a policy so that only entities fulfilling the policy are able to perform the decryption of the message. Symmetrically, a policy-based signature scheme (PBS) assures that only entities fulfilling a given policy are able to generate a valid signature according to the policy. Existing PBC schemes suffer from either inefficiency or lack of strong security arguments. In this paper, we introduce policy-oriented strong security models for PBE and PBS schemes. Then, we present concrete and elegant PBE and PBS schemes from bilinear pairings. Our schemes are not only at least as efficient as existing schemes, but also, and more importantly, provably secure under the defined security models.

keywords: Access Structures, Authorization, Trust Establishment, Bilinear Pairings, Provable Security

1 Introduction

In large-scale open environments like the Internet, many interactions may occur between entities from different security domains without pre-existing trust relationships. Such interactions would not be secure without the establishment of a sufficient level of mutual trust between the communicating entities expressed in terms of compliance with a specific policy. Further, exchange of sensitive resources may also need to be carefully protected through clear and concise authorization policies. An increasingly popular approach for both trust establishment and authorization consists of using policies fulfilled by digital credentials. A digital credential is basically the signature of a specific trusted authority (credential issuer) on a certain assertion about a specific entity (credential owner). It describes one or multiple properties of the entity that are validated by the trusted authority.

The concept of *policy-based cryptography* (PBC), recently formalized in the literature, appears as a promising paradigm for the enforcement of trust establishment and authorization policies. It allows performing cryptographic operations with respect to policies formalized as monotone Boolean expressions. In PBC, a policy involves conjunctions and disjunctions of conditions, where each condition is associated to a specific credential issued by a trusted authority. An entity fulfills a policy if and only if it has access to a set of credentials associated to the logical combination of conditions defined by the policy. Intuitively, a *policy-based encryption* scheme (PBE) allows encrypting a message according to a policy so that only entities fulfilling the policy are able to perform the decryption of the message. Symmetrically, a *policy-based signature* scheme (PBS) assures that only entities fulfilling a given policy are able to generate a valid signature according to the policy.

For an illustration of PBC, consider the following scenario: a web server provides medical information to both doctors who are members of a specific medical association (MA) and patients registered with a specific medical center (MC). When a client sends a request for a specific web page, the server may use a PBE scheme to encrypt the requested web page so that the client can only decrypt it if he has either a doctor's membership credential issued by MA or a patient's registration credential issued by MC. The PBE scheme thus allows the web server to encrypt the requested web page according to the authorization policy 'doctor-member-MA OR patient-registered-MC'. Symmetrically, a patient registered to MC may use his MC credential to sign his request according to the server's policy using a PBS scheme. The validity of the generated signature proves that the client is compliant with the server's policy. Note that, in both scenarios, the web server may not know whether the client is a doctor or a patient. This privacy property will be called credential ambiguity.

The fact that a cryptographic scheme withstood cryptanalytic attacks for several years has been considered, for a long time, as a kind of security validation. Since several schemes have taken a long time before being totally broken, the lack of cryptanalytic attacks or the provision of intuitive or heuristic security arguments are not considered as security validations anymore. The concept of provable security consists of using reductionist proofs as a validation procedure for cryptographic schemes [4, 5, 20]. Concretely, in order to validate a cryptographic protocol, one should first precisely state the security notion he wants the protocol to achieve. This statement results in the definition of intractability for an adversarial goal under a specific attack scenario. A reductionist security proof then is used to show that the intractability of a well-studied mathematical problem can be reduced to the security of the proposed cryptographic protocol with respect to the defined attack model. Existing PBC schemes are either inefficient [14], not supported by formal security models and proofs [2] or supported by weak security arguments [1, 9]. In this paper, our contributions are twofold: on one hand, we introduce policy-oriented strong security models for both PBE and PBS schemes. On the other hand, we present elegant and relatively efficient PBE and PBS schemes and prove their security with respect to the defined attack models. Our PBC primitives are based on bilinear pairings over elliptic curves. Our reductionist security proofs are based on the intractability of well-studied Diffie-Hellman problems [24] while relying on the random oracle model [6].

The paper is organized as follows: we describe our policy model in Section 2. In Section 3, we give a formal definition for PBE schemes, then we define a strong security model for message confidentiality: indistinguishability against adaptive chosen ciphertext attacks. In Section 4, we present a provably secure PBE scheme from bilinear pairings. Our work on PBE schemes owes much to the work on identity-based encryption schemes (IBE) presented in [8] and [12]. We point out that an IBE scheme could be viewed as a PBE scheme for which policies are limited to a single credential issued by a single trusted authority (private key generator). In Section 5, we give a formal definition for PBS schemes. Then, we consider two related security properties: we first define a model for signature unforgeability: existential unforgeability against adaptive chosen message attacks. We then define credential ambiguity against chosen message attacks, which ensures that a valid signature with respect to a given policy does not provide any information about the specific credentials used for its generation. In Section 6, we present a provably secure PBS scheme from bilinear pairings. Our work on PBS schemes is inspired by the work on identity-based ring signatures [19, 25]. Indeed, we use ring structures to deal with disjunctions of conditions within the policy according to which a signature is generated. We discuss related work in Section 7 before concluding in Section 8.

2 Policy Model

Consider a set of trusted authorities denoted $\mathcal{T} = \{TA_1, \ldots, TA_N\}$, where the public key of TA_κ is denoted R_κ and the corresponding master (private) key is denoted s_κ. Each trusted authority may be asked by an entity to issue a credential corresponding to a certain assertion denoted A. The latter may convey information about one or more attributes, properties, and/or capabilities related to the entity such as citizenship, age, group membership, licenses, role within an organization, etc. As the representation of assertions is out of the scope of this paper, they will be henceforth simply encoded as binary strings i.e. $A \in \{0,1\}^*$. Whenever a trusted authority TA_κ is asked by an entity to sign a given assertion A, it first checks the validity of A. If A is valid, then TA_κ executes a credential generation algorithm and returns a credential denoted $\varsigma(R_\kappa, A)$. Otherwise, TA_κ returns an error message. The process of checking the validity of an assertion is out of the scope of this paper. However, we assume that a trusted authority is 'trusted' for never issuing credentials corresponding

to invalid assertions. Note that upon receiving the credential $\varsigma(R_\kappa, A)$, the entity may check its validity/integrity using TA_κ's public key R_κ.

A policy is formalized as a monotone boolean expression involving conjunctions (denoted $\wedge$) and disjunctions (denoted $\vee$) of credential-based conditions. A credential-based condition is defined through a pair $\langle TA_\kappa, A\rangle$ specifying an assertion $A \in \{0,1\}^*$ and an authority $TA_\kappa \in \mathcal{T}$ that is trusted to check and certify A's validity. An entity fulfills the condition $\langle TA_\kappa, A\rangle$ if and only if it has been issued the credential $\varsigma(R_\kappa, A)$. A policy can be written either in conjunctive normal form (CNF) or in disjunctive normal form (DNF). In order to address the two standard normal forms, a policy denoted *Pol* will be written in a generic form called conjunctive-disjunctive normal form (CDNF [21])

$$Pol = \wedge_{i=1}^{m}[\vee_{j=1}^{m_i}[\wedge_{k=1}^{m_{i,j}}\langle TA_{\kappa_{i,j,k}}, A_{i,j,k}\rangle]], \text{ where } TA_{\kappa_{i,j,k}} \in \mathcal{T} \text{ and } A_{i,j,k} \in \{0,1\}^*$$

Therefore, policies written in CNF are such that $m_{i,j} = 1 \; \wr_{j=1}^{m_i} \wr_{i=1}^{m}$, while policies written in DNF correspond to the case where $m = 1$.

Notation. For the sake of clarity to the reader, we use the notation $\wr_{x=x_1}^{x_2}$ as a shortcut for the sentence 'for x varying from x_1 to x_2'. $\diamond$

Note. The maximum values that the quantities m, $m_i \; \wr_{i=1}^{m}$ and $m_{i,j} \; \wr_{j=1}^{m_i} \wr_{i=1}^{m}$ can take are denoted, respectively, $m_{\vee\wedge} \geq 1, m_\vee \geq 1$ and $m_\wedge \geq 1$. We assume that these upper-bounds are specified during the setup stage of a PBC scheme, and that they are publicly known so that we do not need to explicitly provide them subsequently. $\diamond$

Let $\varsigma_{j_1,\ldots,j_m}(Pol)$ denote the set of credentials $\{\{\varsigma(R_{\kappa_{i,j_i,k}}, A_{i,j_i,k})\}_{k=1}^{m_{i,j_i}}\}_{i=1}^{m}$, for some $j_i \in \{1,\ldots,m_i\} \; \wr_{i=1}^{m}$. An entity fulfills the policy *Pol* if and only if it has access to a set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$ for some set of indices $\{j_1,\ldots,j_m\}$. The set $\varsigma_{j_1,\ldots,j_m}(Pol)$ is called a qualified set of credentials for policy *Pol*.

Note. Our approach offers an advantage over threshold schemes in that our schemes address general access structures including those for which there exists no corresponding (k,n)-threshold representation (for $1 < k < n$) (such structures exist according to Theorem 1 of [7]). Although any threshold structure may be written using only $\wedge$ and $\vee$ operators and thus may match our formalism, we believe that dedicated threshold schemes might handle such structures more efficiently and elegantly than our generic approach. $\diamond$

3 Policy-Based Encryption

In this section, we first provide a formal definition for PBE schemes. Then, we describe a policy-oriented security model for indistinguishability against chosen ciphertext attacks.

3.1 Definition

A PBE scheme is specified by five algorithms: Setup, TA-Setup, CredGen, PolEnc and PolDec which we describe below.

Setup. On input of a security parameter k, this algorithm generates some global information I which specifies the different parameters, groups and public functions that will be referenced by subsequent algorithms. Furthermore, it specifies a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$.

Note. We assume that the global information I is publicly known so that we do not need to explicitly provide it as input to subsequent algorithms. $\diamond$

TA-Setup. Let $\mathcal{T} = \{TA_1, \ldots, TA_N\}$ be a set of trusted authorities. Then, each trusted authority $TA_\kappa \in \mathcal{T}$ runs this algorithm to obtain a master key s_κ and a corresponding public key R_κ.

Note. We assume that a trustworthy value of the public key of each trusted authority included in $\mathcal{T}$ is known by the system participants. At any time, a new trusted authority may join the set $\mathcal{T}$. $\diamond$

CredGen. On input of a trusted authority $TA_\kappa \in \mathcal{T}$, an assertion $A \in \{0,1\}^*$, this algorithm is run by TA_κ to obtain the credential $\varsigma(R_\kappa, A)$.

PolEnc. On input of a message $M \in \mathcal{M}$ and a policy Pol, this algorithm returns a ciphertext $C \in \mathcal{C}$ representing the encryption of M with respect to policy Pol.

PolDec. On input of a ciphertext $C \in \mathcal{C}$, a policy Pol and a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, this algorithm returns either a message $M \in \mathcal{M}$ or $\perp$ (for 'error').

The algorithms described above have to satisfy the following consistency constraint:

$$C = \mathsf{PolEnc}(M, Pol) \;\Rightarrow\; \mathsf{PolDec}(C, Pol, \varsigma_{j_1,\ldots,j_m}(Pol)) = M$$

Note. We denote by $\varphi_{j_1,\ldots,j_m}(C, Pol)$ the information from the ciphertext C and the policy Pol that is required to correctly perform the decryption of C with respect to Pol using the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$. A concrete example is given in Section 4. $\diamond$

3.2 Chosen Ciphertext Security

Indistinguishability against adaptive chosen ciphertext attacks for PBE schemes is defined in terms of an interactive game, played between a challenger and an adversary. The game consists of five stages: Setup, Queries-1, Challenge, Queries-2 and Guess which we describe below.

Setup. On input of a security parameter k, the challenger first runs algorithm Setup to obtain the system global information I. Then, the challenger runs algorithm TA-Setup once or multiple times to obtain a set of trusted authorities $\mathcal{T} = \{TA_1, \ldots, TA_N\}$. Finally, the challenger gives to the adversary the global information I as well as the public keys of the different trusted authorities included in $\mathcal{T}$.

Queries-1. The adversary performs a polynomial number of oracle queries adaptively i.e. each query may depend on the replies to the previously performed queries.

Challenge. Once the adversary decides that Queries-1 is over, it gives the challenger two equal length messages M_0, M_1 and a policy Pol_{ch} on which it wishes to be challenged. The challenger picks at random $b \in \{0,1\}$, then runs algorithm PolEnc on input of the tuple (M_b, Pol_{ch}), and finally returns the resulting ciphertext C_{ch} to the adversary.

Queries-2. The adversary performs again a polynomial number of adaptive oracle queries.

Guess. The adversary outputs a guess b', and wins the game if $b = b'$.

The oracles that the adversary may query during Queries-1 and Queries-2 are defined as follows:

- **CredGen-O**. On input of a trusted authority $TA_\kappa \in \mathcal{T}$ and an assertion $A \in \{0,1\}^*$, run algorithm CredGen on input of the tuple (TA_κ, A), and return the resulting credential $\varsigma(R_\kappa, A)$.
- **PolDec-O**. On input of $C \in \mathcal{C}$, a policy Pol and a set of indices $\{j_1, \ldots, j_m\}$, first run algorithm CredGen multiple times to obtain the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, then run algorithm PolDec on input of the tuple $(C, Pol, \varsigma_{j_1,\ldots,j_m}(Pol))$, and return the resulting output.

The oracle queries made by the adversary during Queries-1 and Queries-2 are subject to two restrictions. On one hand, the adversary is not allowed to obtain a qualified set of credentials for the

challenge policy Pol_{ch}. On the other hand, he is not allowed to perform a query to oracle PolDec-O on a tuple $(C, Pol, \{j_1, \ldots, j_m\})$ such that $\varphi_{j_1,\ldots,j_m}(C, Pol) = \varphi_{j_1,\ldots,j_m}(C_{\text{ch}}, Pol_{\text{ch}})$.

The game described above is denoted IND-Pol-CCA. A formal definition of chosen ciphertext security for PBE schemes is given below. As usual, a real function g is said to be negligible if $g(k) \leq \frac{1}{f(k)}$ for any polynomial f.

Definition 1. *The advantage of an adversary $\mathcal{A}$ in the* IND-Pol-CCA *game is defined to be the quantity $Adv_{\mathcal{A}} = |Pr[b = b'] - \frac{1}{2}|$. A* PBE *scheme is* IND-Pol-CCA *secure if no probabilistic polynomial time adversary has a non-negligible advantage in the* IND-Pol-CCA *game.*

Note. Our IND-Pol-CCA model could be viewed as an extension to the policy-based setting of the IND-ID-CCA model defined in [8]. In IND-ID-CCA, the adversary is not allowed to make decryption queries on the challenge tuple $(C_{\text{ch}}, ID_{\text{ch}})$. In the policy-based setting, for an encrypted message with respect to a policy with disjunctions, there is more than one possible qualified set of credentials that can be used to perform the decryption. That is, forbidding the adversary from making decryption queries on the challenge tuple $(C_{\text{ch}}, Pol_{\text{ch}})$ is not sufficient anymore. In fact, we may have tuples such that $(C, Pol) \neq (C_{\text{ch}}, Pol_{\text{ch}})$ while $\varphi_{j_1,\ldots,j_m}(C, Pol) = \varphi_{j_1,\ldots,j_m}(C_{\text{ch}}, Pol_{\text{ch}})$. Decryption queries on such tuples should then be forbidden as well. $\diamond$

4 Our PBE Scheme

In this section, we first describe our PBE scheme. Then, we discuss its efficiency and analyze its security in the random oracle model.

4.1 Description

Before describing our PBE scheme, we define algorithm BDH-Setup as follows:

BDH-Setup. On input of a security parameter k, generate a tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P)$ where the map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear pairing, $(\mathbb{G}_1, +)$ and $(\mathbb{G}_2, *)$ are two Gap Diffie-Hellman groups of the same order q, and P is a random generator of $\mathbb{G}_1$. The generated parameters are such that the Bilinear Diffie-Hellman Problem (denoted BDHP) is hard.

Note. We recall that a bilinear pairing satisfies the following three properties: (1) Bilinear: for $Q, Q' \in \mathbb{G}_1$ and for $a, b \in \mathbb{Z}_q^*$, $e(a \cdot Q, b \cdot Q') = e(Q, Q')^{ab}$, (2) Non-degenerate: $e(P, P) \neq 1$ and therefore it is a generator of $\mathbb{G}_2$, (3) Computable: there exists an efficient algorithm to compute $e(Q, Q')$ for all $Q, Q' \in \mathbb{G}_1$. $\diamond$

Note. BDHP is defined as follows: on input of a tuple $(P, a \cdot P, b \cdot P, c \cdot P)$ for randomly chosen $a, b, c \in \mathbb{Z}_q^*$, compute the value $e(P, P)^{abc}$. The hardness of BDHP can be ensured by choosing groups on supersingular elliptic curves or hyperelliptic curves over finite fields and deriving the bilinear pairings from Weil or Tate pairings. The hardness of BDHP implies the hardness of the so called Computational Diffie-Hellman Problem (denoted CDHP) which is defined as follows: on input of a tuple $(P, a \cdot P, b \cdot P)$ for randomly chosen $a, b \in \mathbb{Z}_q^*$, compute the value $ab \cdot P$. As we merely apply these mathematical primitives in this paper, we refer for instance to [15, 24] for further details. $\diamond$

Our PBE scheme consists of the algorithms described below.

Setup. On input of a security parameter k, do the following: (1) Run algorithm BDH-Setup to obtain a tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P)$, (2) Let $\mathcal{M} = \{0,1\}^{n-n_0}$ and $\mathcal{C} = \mathbb{G}_1 \times (\{0,1\}^n)^*$ (for some $n, n_0 \in \mathbb{N}^*$ such that $n_0 \ll n$), (3) Define three hash functions: $H_0 : \{0,1\}^* \rightarrow \mathbb{G}_1$, $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^n$, (4) Let $I = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, n_0, H_0, H_1, H_2)$.

TA-Setup. Let $\mathcal{T} = \{TA_1, \ldots, TA_N\}$ be a set of trusted authorities. Each trusted authority $TA_\kappa \in \mathcal{T}$ picks at random a secret master key $s_\kappa \in \mathbb{Z}_q^*$ and publishes the corresponding public key $R_\kappa = s_\kappa \cdot P$.

CredGen. On input of $TA_\kappa \in \mathcal{T}$ and $A \in \{0,1\}^*$, this algorithm outputs $\varsigma(R_\kappa, A) = s_\kappa \cdot H_0(A)$.

PolEnc. On input of message $M \in \mathcal{M}$ and policy Pol, do the following:

1. Pick at random $M_i \in \{0,1\}^{n-n_0}$ $\wr_{i=1}^{m-1}$, then set $M_m = M \oplus (\oplus_{i=1}^{m-1} M_i)$
2. Pick at random $t_i \in \{0,1\}^{n_0}$ $\wr_{i=1}^{m}$
3. Compute $r = H_1(M_1 \| \ldots \| M_m \| t_1 \| \ldots \| t_m)$, then compute $U = r \cdot P$
4. Compute $\pi_{i,j} = \prod_{k=1}^{m_{i,j}} e(R_{\kappa_{i,j,k}}, H_0(A_{i,j,k}))$ $\wr_{j=1}^{m_i} \wr_{i=1}^{m}$
5. Compute $\mu_{i,j} = H_2(\pi_{i,j}^r \| i \| j)$, then compute $v_{i,j} = (M_i \| t_i) \oplus \mu_{i,j}$ $\wr_{j=1}^{m_i} \wr_{i=1}^{m}$
6. Return $C = (U, [[v_{i,j}]_{j=1}^{m_i}]_{i=1}^{m})$

The intuition behind the encryption algorithm PolEnc is as follows: each conjunction of conditions $\wedge_{k=1}^{m_{i,j}} \langle TA_{\kappa_{i,j,k}}, A_{i,j,k} \rangle$ is first associated to a mask $\mu_{i,j}$ that depends on the different credentials related to the specified conditions. The encrypted message M is split into m random shares $[M_i]_{i=1}^{m}$, then for each index $i \in \{1, \ldots, m\}$, the value $M_i \| t_i$ is associated to the disjunction $\vee_{j=1}^{m_i} \wedge_{k=1}^{m_{i,j}} \langle TA_{\kappa_{i,j,k}}, A_{i,j,k} \rangle$, where t_i is a randomly chosen intermediate key. Each value $M_i \| t_i$ is encrypted m_i times using each of the masks $\mu_{i,j}$. This way, it is sufficient to compute any one of the masks $\mu_{i,j}$ in order to be able to retrieve $M_i \| t_i$. In order to be able to retrieve the encrypted message, an entity needs to retrieve all the shares as well as all the intermediate keys t_i using a set of qualified credentials for policy Pol.

PolDec. On input of ciphertext $C = (U, [[v_{i,j}]_{j=1}^{m_i}]_{i=1}^{m})$, policy Pol, and the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, do the following:

1. Compute $\tilde{\pi}_{i,j_i} = e(U, \sum_{k=1}^{m_{i,j_i}} \varsigma(R_{\kappa_{i,j_i,k}}, A_{i,j_i,k}))$ $\wr_{i=1}^{m}$
2. Compute $\tilde{\mu}_{i,j_i} = H_2(\tilde{\pi}_{i,j_i} \| i \| j_i)$, then compute $(M_i \| t_i) = v_{i,j_i} \oplus \tilde{\mu}_{i,j_i}$ $\wr_{i=1}^{m}$
3. Compute $r = H_1(M_1 \| \ldots \| M_m \| t_1 \| \ldots \| t_m)$
4. If $U = r \cdot P$, then return the message $M = \oplus_{i=1}^{m} M_i$, otherwise return $\perp$

Note. Our PBE scheme is such that the decryption information $\varphi_{j_1,\ldots,j_m}(C = (U, [[v_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}), Pol)$ consists of the value U and the pairs $(v_{i,j_i}, \wedge_{k=1}^{m_{i,j_i}} \langle TA_{\kappa_{i,j_i,k}}, A_{i,j_i,k} \rangle)$ $\wr_{i=1}^{m}$. $\diamond$

Note. As opposed to the scheme provided in [9], chosen ciphertext security is provided by the random value r being a function of the intermediate keys and the encrypted message. If the policy specified by the sender during the encryption is considered as sensitive, it can be hidden from the recipient. In this case, the latter needs to test different combinations of credentials from his set of credentials until retrieving the encrypted message. As in the modified secret splitting scheme of [9], the different intermediate keys can be generated in a way such that only a recipient fulfilling the hidden policy recognizes correct decryptions of the different intermediate keys.

4.2 Consistency and Efficiency

Our PBE scheme satisfies the standard consistency constraint. In fact, thanks to the bilinearity property of bilinear pairings, the following holds

$$\tilde{\pi}_{i,j_i} = e(r \cdot P, \sum_{k=1}^{m_{i,j_i}} s_{\kappa_{i,j_i,k}} \cdot H_0(A_{i,j_i,k})) = \prod_{k=1}^{m_{i,j_i}} e(s_{\kappa_{i,j_i,k}} \cdot P, H_0(A_{i,j_i,k}))^r = \pi_{i,j_i}^r$$

The essential operation in pairing-based cryptography is pairing computation. Although such operation can be optimized as explained in [3], it still have to be minimized. On one hand, our encryption

algorithm PolEnc requires a total of $\sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}$ pairing computations, each one corresponds to a credential-based condition specified by the policy. The same amount of pairing computations is required by the encryption algorithms proposed in [2, 9]. On the other hand, our decryption algorithm PolDec requires a total of m pairing computations (same as in [2]), each one corresponds to a message share concatenated to an intermediate key. Whereas, the decryption algorithm proposed in [9] requires a total of $\sum_{i=1}^{m} m_{i,j_i}$ pairing computations.

Note. In algorithm PolEnc, the value $e(R_{\kappa_{i,j,k}}, H_0(A_{i,j,k}))$ does not depend on the encrypted message M. Thus, it can be pre-computed, cached and used in subsequent encryptions involving the credential-based condition $\langle TA_{\kappa_{i,j,k}}, A_{i,j,k}\rangle$. $\diamond$

Let l_1 denote the bit-length of the bilinear representation of an element of group $\mathbb{G}_1$, then the size of a ciphertext produced by our PBE scheme is equal to $l_1 + (\sum_{i=1}^{m} m_i).n$. Thus, our scheme produces more compact ciphertexts than the scheme proposed in [2] ($l_1 + (\sum_{i=1}^{m} m_i).n + n$) and the one proposed in [9] ($l_1 + (\sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}).n + n$).

4.3 Security

Theorem 1. *Our* PBE *scheme is* IND-Pol-CCA *secure in the random oracle model under the assumption that BDHP is hard.*

Proof. Theorem 1 follows from a sequence of three reduction arguments: (1) In [8], the public-key encryption scheme BasicPub is shown to be IND-CPA secure in the random oracle model under the assumption that BDHP is hard, (2) In [12], the public-key encryption scheme NewBasicPub^{hy} is defined to be the encryption scheme resulting from the application of the Fujisaki-Okamoto transformation [11] to the BasicPub scheme. Result 6 of [12] shows that an IND-CCA attack on NewBasicPub^{hy} can be converted into an IND-CPA attack on BasicPub, (3) Lemma 1 shows that an IND-Pol-CCA attack on our PBE scheme can be converted into an IND-CCA attack on NewBasicPub^{hy}. Lemma 1 is defined below.

Lemma 1. *Let $\mathcal{A}^{\circ}$ be an* IND-Pol-CCA *adversary with advantage $Adv_{\mathcal{A}^{\circ}} \geq \varepsilon$ when attacking our* PBE *scheme. Assume that $\mathcal{A}^{\circ}$ has running time $t_{\mathcal{A}^{\circ}}$ and makes at most q_c queries to oracle* CredGen-O*, q_d queries to oracle* PolDec-O *as well as q_0 queries to oracle H_0. Then, there exists an* IND-ID-CCA *adversary $\mathcal{A}^{\bullet}$ the advantage of which, when attacking the* NewBasicPub^{hy} *scheme, is such that $Adv_{\mathcal{A}^{\bullet}} \geq F(q_c, q_d, q_0, N, m_{\vee\wedge}, m_{\vee}, m_{\wedge}).\varepsilon$. Its running time is $t_{\mathcal{A}^{\bullet}} = O(t_{\mathcal{A}^{\circ}})$.*

Proof of Lemma 1 is given in Appendix A.

Note. Computing $F(q_c, q_d, q_0, N, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ relies on computing the quantity $\Upsilon(X, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$, which is defined to be the total number of 'minimal' policies written in CDNF, given the upper-bounds $(m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ and X possible credential-based conditions. Computing $\Upsilon(X, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ is similar, but not exactly the same as the problems of computing the number of monotone boolean functions of n variables (Dedekind's Problem [17]) and computing the number of antichains on a set $\{1, \ldots, n\}$ [16]. As opposed to these problems, the order of the terms must be taken into consideration when dealing with our policies. This is a typical, yet interesting, 'counting' problem. However, due to space limitations, we do not elaborate more on the details. $\diamond$

$\square$

5 Policy-Based Signature

In this section, we first provide a formal definition for PBS schemes. Then, we describe security models for existential unforgeability and credential ambiguity against chosen message attacks.

5.1 Definition

A PBS scheme is specified by five algorithms: Setup, TA-Setup, CredGen, PolSig and PolVrf which we describe below.

Setup. On input of a security parameter k, this algorithm generates some global information I which specifies the different parameters, groups and public functions that will be referenced by subsequent algorithms. Furthermore, it specifies a message space $\mathcal{M}$ and a signature space $\mathcal{S}$.

TA-Setup. As for PBE schemes (Section 3).

CredGen. As for PBE schemes (Section 3).

PolSig. On input of a message $M \in \mathcal{M}$, a policy Pol and a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, this algorithm returns a signature $\sigma \in \mathcal{S}$ representing the signature on M according to policy Pol.

PolVrf. On input of a message $M \in \mathcal{M}$, a signature $\sigma \in \mathcal{S}$ and a policy Pol, this algorithm returns $\top$ (for true) if σ is a valid signature on M according to policy Pol. Otherwise, it returns $\perp$ (for false).

The algorithms described above have to satisfy the following consistency constraint:

$$\sigma = \mathsf{PolSig}(M, Pol, \varsigma_{j_1,\ldots,j_m}(Pol)) \;\Rightarrow\; \mathsf{PolVrf}(M, \sigma, Pol) = \top$$

5.2 Existential Unforgeability

Existential unforgeability against chosen message attacks for PBS schemes is defined in terms of an interactive game (denoted EUF-Pol-CMA), played between a challenger and an adversary. The game consists of three stages: Setup, Probing and Guess which we describe below.

Setup. On input of a security parameter k, the challenger first runs algorithm Setup to obtain the system global information I. Then, the challenger runs algorithm TA-Setup once or multiple times to obtain a set of trusted authorities $\mathcal{T} = \{TA_1, \ldots, TA_N\}$. Finally, the adversary gives to the adversary the global information I as well as the public keys of the different trusted authorities included in $\mathcal{T}$.

Probing. The adversary performs a polynomial number of oracle queries adaptively.

Forge. Once the adversary decides that Probing is over, it outputs a message M_f, a policy Pol_f and a signature σ_f. The adversary wins the game if $\mathsf{PolVrf}(M_\mathrm{f}, \sigma_\mathrm{f}, Pol_\mathrm{f}) = \top$.

The oracles that the adversary may query during Probing are defined as follows:

- **CredGen-O**. As for the IND-Pol-CCA game (Section 3).
- **PolSig-O**. On input of a message M and a policy Pol, first run algorithm CredGen once or multiple times to obtain a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$ for policy Pol, then run algorithm PolSig on input the tuple $(M, Pol, \varsigma_{j_1,\ldots,j_m}(Pol))$ and return the resulting output.

The oracle queries performed by the adversary during Probing are subject to some restrictions. In fact, the adversary is not allowed to obtain (through queries to oracle CredGen-O) a set of credentials fulfilling the forgery policy Pol_f. Besides, it is natural to assume that the adversary does not perform a query on the tuple $(M_\mathrm{f}, Pol_\mathrm{f})$ to oracle PolSig-O.

Definition 2. *The advantage of an adversary $\mathcal{A}$ in the* EUF-Pol-CMA *game is defined to be the quantity $Adv_\mathcal{A} = Pr[\mathcal{A}\ wins]$. A* PBS *scheme is* EUF-Pol-CMA *secure if no probabilistic polynomial time adversary has a non-negligible advantage in the* EUF-Pol-CMA *game.*

5.3 Credential Ambiguity

Credential ambiguity against chosen message attacks for PBS schemes is defined in terms of an interactive game (denoted CrA-Pol-CMA), played between a challenger and an adversary. The game consists of three stages: Setup, Challenge and Guess which we describe below.

Setup. As in the EUF-Pol-CMA game. Here, the adversary is additionally given the master keys of the different trusted authorities.

Challenge. The adversary chooses a message M_{ch} and a policy Pol_{ch} on which he wishes to be challenged. The challenger does the following: (1) Pick at random $j_i^{\text{ch}} \in \{1,\ldots,m_i\}$ $\wr_{i=1}^{m}$, (2) Run algorithm CredGen m times to obtain the qualified set of credentials $\varsigma_{j_1^{\text{ch}},\ldots,j_m^{\text{ch}}}(Pol_{\text{ch}})$, (3) Run algorithm PolSig on input the tuple $(M_{\text{ch}}, Pol_{\text{ch}}, \varsigma_{j_1^{\text{ch}},\ldots,j_m^{\text{ch}}}(Pol_{\text{ch}}))$ and return the resulting output σ_{ch} to the adversary.

Guess. The adversary outputs a tuple $(j_1,\ldots,j_m)$, and wins the game if $(j_1^{\text{ch}},\ldots,j_m^{\text{ch}}) = (j_1,\ldots,j_m)$.

Definition 3. *The advantage of an adversary $\mathcal{A}$ in the* CrA-Pol-CMA *game is defined to be the quantity $Adv_{\mathcal{A}} = \mathsf{Max}_i\{|Pr[j_i = j_i^{ch}] - \frac{1}{m_i}|\}$. A* PBS *scheme is* CrA-Pol-CMA *secure if no probabilistic polynomial time adversary has a non-negligible advantage in the* CrA-Pol-CMA *game.*

6 Our PBS Scheme

In this section, we first describe our PBS scheme. Then, we discuss its efficiency and analyze its security in the random oracle model.

6.1 Description

Our PBS scheme consists of the algorithms described below.

Setup. On input of a security parameter k, do the following: (1) Run algorithm BDH-Setup on input k to generate output $(q,\mathbb{G}_1,\mathbb{G}_2,e,P)$, (2) For some chosen $n \in \mathbb{N}^*$, let $\mathcal{M} = \{0,1\}^n$ and let $\mathcal{S} = (\mathbb{G}_2)^* \times \mathbb{G}_1$, (3) Define two hash functions: $H_0 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_4 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$, (4) Let $I = (q,\mathbb{G}_1,\mathbb{G}_2,e,P,n,H_0,H_4)$.

TA-Setup. As for our PBE Scheme (Section 4).

CredGen. As for our PBE Scheme (Section 4).

PolSig. On input of a message $M \in \mathcal{M}$, a policy Pol and a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, do the following:

1. Pick at random $Y_i \in \mathbb{G}_1$, then compute $x_{i,j_i+1} = e(P,Y_i)$ $\wr_{i=1}^{m}$
2. For $l = j_i+1,\ldots,m_i,1,\ldots,j_i-1 \bmod(m_i+1)$ $\wr_{i=1}^{m}$, do the following:
   (a) Compute $\tau_{i,l} = \prod_{k=1}^{m_{i,l}} e(R_{\kappa_{i,l,k}}, H_0(A_{i,l,k}))$
   (b) Pick at random $Y_{i,l} \in \mathbb{G}_1$, then compute $x_{i,l+1} = e(P,Y_{i,l}) * \tau_{i,l}^{H_4(M\|x_{i,l}\|m\|i\|l)}$
3. Compute $Y_{i,j_i} = Y_i - H_4(M\|x_{i,j_i}\|m\|i\|j_i) \cdot (\sum_{k=1}^{m_{i,j_i}} \varsigma(R_{\kappa_{i,j_i,k}}, A_{i,j_i,k}))$ $\wr_{i=1}^{m}$
4. Compute $Y = \sum_{i=1}^{m}\sum_{j=1}^{m_i} Y_{i,j}$
5. Return $\sigma = ([[x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y)$

The intuition behind algorithm PolSig is as follows: each conjunction of credential-based conditions $\wedge_{k=1}^{m_{i,j}} \langle TA_{\kappa_{i,j,k}}, A_{i,j,k}\rangle$ is associated to a tag $\tau_{i,j}$. For each index i, the set of tags $\{\tau_{i,j}\}_{j=1}^{m_i}$ is equivalent to a set of ring members. The signature key of the ring member corresponding to the tag $\tau_{i,j}$ consists

of the credentials $\{\varsigma(R_{\kappa_{i,j,k}}, A_{i,j,k})\}_{k=1}^{m_{i,j}}$. Thus, the generated signature corresponds to a set of ring signatures which validity can be checked using the 'glue' value Y.

PolVrf. Let $\sigma = ([[x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y)$ be a signature on message M according to policy Pol. To check the validity of σ, do the following:

1. Compute $z_1 = \prod_{i=1}^{m}[\prod_{j=1}^{m_i} x_{i,j}]$
2. Compute $\tau_{i,j} = \prod_{k=1}^{m_{i,j}} e(R_{\kappa_{i,j,k}}, H_0(A_{i,j,k}))\ \wr_{j=1}^{m_i} \wr_{i=1}^{m}$
3. Compute $z_2 = e(P,Y) * \prod_{i=1}^{m}\prod_{j=1}^{m_i} \tau_{i,j}^{H_4(M\|x_{i,j}\|m\|j\|i)}$
4. If $z_1 = z_2$, then return $\top$, otherwise return $\perp$

6.2 Consistency and Efficiency

Our PBS scheme satisfies the standard consistency constraint. In fact, on one hand, the following statement holds

$$\tau_{i,j}^{H_4(M\|x_{i,j}\|m\|i\|j)} = x_{i,j+1} * e(P,Y_{i,j})^{-1} \text{ (where } x_{i,m_i+1} = x_{i,1})\ \wr_{j=1}^{m_i} \wr_{i=1}^{m}$$

On the other hand, let $\lambda = e(P,Y)$, then the following holds

$$z_2 = \lambda * \prod_{i=1}^{m}[\prod_{j=1}^{m_i} \tau_{i,j}^{H_4(M\|x_{i,j}\|m\|i\|j)}] = \lambda * \prod_{i=1}^{m}[\prod_{j=1}^{m_i-1} x_{i,j+1} * e(P,Y_{i,j})^{-1} * x_{i,1} * e(P,Y_{i,m_i})^{-1}]$$
$$= \lambda * \prod_{i=1}^{m}[\prod_{j=1}^{m_i} x_{i,j} * \prod_{j=1}^{m_i} e(P,Y_{i,j})^{-1}] = \lambda * [\prod_{i=1}^{m}\prod_{j=1}^{m_i} x_{i,j}] * [e(P, \sum_{i=1}^{m}\sum_{j=1}^{m_i} Y_{i,j})]^{-1} = \lambda * z_1 * \lambda^{-1}$$

Our signature algorithm PolSig requires a total of $\sum_{i=1}^{m} m_i + \sum_{i=1}^{m}\sum_{j\neq j_i} m_{i,j}$ pairing computations, whereas our verification algorithm Polvrf requires a total of $1 + \sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}$ pairing computations. Up to $\sum_{i=1}^{m}\sum_{j\neq j_i} m_{i,j}$ in algorithm PolSig and up to $\sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}$ in algorithm PolVrf can be pre-computed, cached and used in subsequent interactions.

Let l_i denote the bit-length of the bilinear representation of an element of group $\mathbb{G}_i\ \wr_{i=1,2}$, then the bit-length of a signature produced by our PBS scheme is equal to $(\sum_{i=1}^{m} m_i).l_2 + l_1$.

Note. A PBS scheme can be constructed from the ID-based ring signature proposed in [25]. However, the way this ring signature was constructed does not allow generating a single 'glue' value for the policy according to which the signature has to be generated. $\diamond$

6.3 Security

Theorem 2. *Our* PBS *scheme is* EUF-Pol-CMA *secure in the random oracle model under the assumption that CDHP is hard.*

Proof. Theorem 2 follows from Lemma 2 which we define below.

Lemma 2. *Let $\mathcal{A}^{\circ}$ be an* EUF-Pol-CMA *adversary with advantage $Adv_{\mathcal{A}^{\circ}} \geq \varepsilon$ when attacking the* PBS *scheme, where ε is a non-negligible function. Assume that adversary $\mathcal{A}^{\circ}$ has running time $t_{\mathcal{A}^{\circ}}$ and makes at most q_c queries to oracle* CredGen-O*, q_s queries to oracle* PolSig-O*, q_0 queries to oracle H_0 and q_4 queries to oracle H_4. Then, there exists an adversary $\mathcal{A}^{\bullet}$ the advantage of which, when attacking* CDHP*, is such that $Adv_{\mathcal{A}^{\bullet}} \geq 9/(100 q_0^{m_{\vee\wedge}m_{\vee}} \sum_{l=1}^{m_{\vee}} l!\binom{m_{\vee}}{l})$. For $q \geq Max\{2m_{\vee\wedge}m_{\vee}, 2m_{\vee\wedge}q_s q_4\}$ and $\varepsilon \leq 32(q_4 + 1 - m_{\vee\wedge}m_{\vee})/q$, its running time is such that $t_{\mathcal{A}^{\bullet}} \leq (32q_4 + 4)t_{\mathcal{A}^{\circ}}/\varepsilon$.*

Proof of Lemma 2 follows the method described in [13], which is based on the oracle replay technique [20]. Informally, by a polynomial replay of the attack with different random oracles, we allow the attacker to forge two signatures that are related so that the attacker is able to solve the underlying hard problem (CDHP). The details of our proof are given in Appendix B.

□

Theorem 3. *Our* PBS *scheme is* CrA-Pol-CMA *secure in the random oracle model.*

Proof. Let M_{ch} be the message and $\sigma_{\text{ch}} = ([x_{i,j}^{\text{ch}}]_{j=1}^{m_i}]_{i=1}^{m}, Y^{\text{ch}})$ be the signature which the adversary is challenged on in the CrA-Pol-CMA game. Our PBS scheme is such that the following hold

1. $x_{i,j}^{\text{ch}} = e(P, Y_{i,j-1}) * \tau_{i,j-1}^{H_4(M_{\text{ch}} \| x_{i,j-1}^{\text{ch}} \| m \| i \| j-1)} \; \wr_{j \neq j_i^{\text{ch}}+1}$, $x_{i,j_i^{\text{ch}}+1}^{\text{ch}} = e(P, Y_i) \quad \wr_{i=1}^{m}$
2. $Y^{\text{ch}} = \sum_{i=1}^{m} [\sum_{j \neq j^{\text{ch}}} Y_{i,j} + Y_i - H_4(M_{\text{ch}} \| x_{i,j_i^{\text{ch}}}^{\text{ch}} \| m \| i \| j_i^{\text{ch}}) \cdot (\sum_{k=1}^{m_{i,j_i^{\text{ch}}}} \varsigma(R_{\kappa_{i,j_i^{\text{ch}},k}}, A_{i,j_i^{\text{ch}},k}))]$

Since Y_i and $Y_{i,j-1}$ are chosen at random from $\mathbb{G}_1$, and H_4 is assumed to be a random oracle, we have that $x_{i,j}^{\text{ch}}$ and Y^{ch} are uniformly distributed in $\mathbb{G}_2$ and $\mathbb{G}_1$ respectively. If $(j_1, \ldots, j_m)$ is the tuple output by the adversary in the CrA-Pol-CMA game, then we have $Pr[j_i = j_i^{\text{ch}}] = 1/m_i \; \wr_{i=1}^{m}$.

□

7 Related Work

The problem of performing encryption with respect to policies formalized as monotone boolean expressions has been addressed in various areas. In [10], the authors show how to perform encryptions according to disjunctions and conjunctions of decryption keys generated by multiple trusted authorities. However, their solution remains restricted to a limited number of disjunctions. In [21], the author further pursues the ideas discussed in [10] and presents an elegant and efficient mechanism to perform encryption according to arbitrary combinations of keys, yet generated by a single trusted authority. Our work on PBE could be seen as an extension of [21] in the sense that we use the same policy model while dealing with multiple trusted authorities. Furthermore, as opposed to our work, the schemes proposed in [10, 21] are not supported by security models and proofs.

Automated trust negotiation (ATN) allows regulating the flow of security/privacy-sensitive resources during trust establishment through the definition of disclosure policies [23]. One of the major problems in ATN is called the cyclic policy interdependency problem which occurs when a communication party is obliged to be the first to reveal a sensitive credential to the other. In [18], the authors model the cyclic policy interdependency problem as a 2-party secure function evaluation (SFE) and propose oblivious signature-based envelopes (OSBE) for efficiently solving the FSE problem. They describe a one-round OSBE scheme based on ID-based cryptography. Our work on PBE could be seen as a generalization of their ID-based OSBE scheme in the sense that the latter corresponds to encrypting a message with respect to a policy fulfilled by a single credential.

In [14], the authors introduce the notion of hidden credentials which is equivalent to PBE in that the ability to read a sensitive resource is contingent on having been issued the required credentials. Compared to OSBE, hidden credentials deal with complex policies expressed as monotone boolean expressions. They use onion-like encryptions and multiple encryptions to deal, respectively, with conjunctions and disjunctions of credentials. Their approach remains inefficient in terms of both computational costs and bandwidth consumption (ciphertext size), especially when authorization

structures become very complex. In [9], the authors propose a solution to improve decryption efficiency as well as policy concealment when implementing hidden credentials with sensitive policies. They prove the chosen ciphertext security of their solution while relying on the security models defined in [8] for ID-based encryption schemes. As opposed to their approach, ours consists of defining a policy-oriented chosen ciphertext security model, which enables us to provide more formal reductionist security analysis.

An escrow-free public-key encryption scheme supporting cryptographic workflow is presented in [1]. The proposed scheme could be called a public-key policy-based encryption scheme as the ability to read a sensitive resource requires not only the possession of a qualified set of credentials but also the knowledge of a specific private key. Formal security models are defined to support the presented scheme. The recipient security model considered in [1] corresponds to indistinguishability against chosen plaintext attacks for PBE schemes, whereas our security analysis considers the stronger chosen ciphertext attacks. The encryption scheme of [1] and the one of [9] consider policies formalized as monotone boolean expressions represented as general conjunctions and disjunctions of atomic terms. The size of the resulting ciphertexts depends linearly on the number of these terms, whereas the normal forms used in our approach substantially reduce the size of the produced ciphertexts.

In [2], the authors show how the policy-based cryptographic primitives can be used to enforce policies with respect to the data minimization principle according to which only strictly necessary information should be collected for a given purpose. In this paper, we provide a further refinement of the formal definitions given in [2]. While the security analysis in [2] is intuitive, we define in this paper new security models for PBC primitives and give reductionist security arguments for our PBC schemes. Furthermore, our PBE scheme produces ciphertexts with shorter size compared to the scheme presented in [2].

As opposed to encryption, there is lesser emphasis in the literature on the specific problem of generating pairing-based signatures with respect to policies formalized as monotone boolean functions. PBS schemes may have interesting application in proof-based authorization systems [22]. Our PBS scheme could be viewed as a slight modification of the scheme presented in [2]. In contrast with [2], our scheme is supported by formal security models and proofs. The latter follows the technique presented in [13] to prove the security of the ID-based ring signature of [25]. Note finally that our proof can be easily adapted to prove the security of the ID-based ring signature of [19].

8 Conclusion

The concept of PBC allows to perform cryptographic operations with respect to policies formalized as monotone boolean expressions. Such operations have interesting applications in encryption-based and proof-based authorization systems as well as in trust establishment and negotiation. Various PBC schemes proposed so far in the literature suffer from either inefficiency or lack of strong security arguments. In this paper, we first introduced policy-oriented strong security models for both PBE and PBS schemes. Then, we presented concrete PBE and PBS schemes and proved their security in the random oracle model. Future research directions include further improvement on the efficiency of PBE and PBS schemes, provision of tighter security reductions, development of new PBC primitives and investigation of the deployment of PBC schemes in real-world applications.

References

1. S.S. Al-Riyami, J. Malone-Lee, and N.P. Smart. Escrow-free encryption supporting cryptographic workflow. Cryptology ePrint Archive, Report 2004/258, 2004. http://eprint.iacr.org/.
2. W. Bagga and R. Molva. Policy-based cryptography and applications. In *Proceedings of Financial Cryptography and Data Security (FC'05)*, volume 3570 of *LNCS*, pages 72–87. Springer-Verlag, 2005.
3. P. Barreto, H. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368. Springer-Verlag, 2002.
4. M. Bellare. Practice-oriented provable-security. In *ISW '97: Proceedings of the First International Workshop on Information Security*, pages 221–231. Springer-Verlag, 1998.
5. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 26–45. Springer-Verlag, 1998.
6. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM Press, 1993.
7. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 27–35, New York, NY, USA, 1990. Springer-Verlag New York, Inc.
8. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229. Springer-Verlag, 2001.
9. Robert Bradshaw, Jason Holt, and Kent Seamons. Concealing complex policies with hidden credentials. Cryptology ePrint Archive, Report 2004/109, 2004. http://eprint.iacr.org/.
10. L. Chen, K. Harrison, D. Soldera, and N. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *Proceedings of the International Conference on Infrastructure Security*, pages 260–275. Springer-Verlag, 2002.
11. E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In *Public Key Cryptography*, pages 53–68, 1999.
12. David Galindo. Boneh-franklin identity based encryption revisited. To appear in Proceedings of 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005).
13. J. Herranz. A formal proof of security of zhang and kim's id-based ring signature scheme. In *WOSIS'04*, pages 63–72. INSTICC Press, 2004. ISBN 972-8865-07-4.
14. J. Holt, R. Bradshaw, K. E. Seamons, and H. Orman. Hidden credentials. In *Proc. of the 2003 ACM Workshop on Privacy in the Electronic Society*. ACM Press, 2003.
15. A. Joux. The weil and tate pairings as building blocks for public key cryptosystems. In *Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 20–32. Springer-Verlag, 2002.
16. J. Kahn. Entropy, independent sets and antichains: a new approach to dedekind's problem. In *Proc. Amer. Math. Soc. 130*, pages 371–378, 2002.
17. D. Kleitman. On dedekind's problem: the number of monotone boolean functions. In *Proc. Amer. Math. Soc. 21*, pages 677–682, 1969.
18. N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *Proceedings of the 22nd annual symposium on Principles of distributed computing*, pages 182–189. ACM Press, 2003.
19. C. Lin and T. Wu. An identity-based ring signature scheme from bilinear pairings. Cryptology ePrint Archive, Report 2003/117, 2003. http://eprint.iacr.org/.
20. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(3):361–396, 2000.
21. N. Smart. Access control using pairing based cryptography. In *Proceedings CT-RSA 2003*, pages 111–121. Springer-Verlag LNCS 2612, April 2003.
22. A. Hess T. Barlow and K. E. Seamons. Exploiting hierarchical identity-based encryption for access control to pervasive computing information. Technical Report CMU-CS-04-172, October 2004.
23. W. Winsborough, K. Seamons, and V. Jones. Automated trust negotiation. Technical Report TR-2000-05, Raleigh, NC, USA, 24 2000.
24. Y. Yacobi. A note on the bilinear diffie-hellman assumption. Cryptology ePrint Archive, Report 2002/113, 2002. http://eprint.iacr.org/.
25. F. Zhang and K. Kim. ID-based blind signature and ring signature from pairings. In *ASIACRYPT*, pages 533–547. Springer-Verlag LNCS 2501, 2002.

A Proof of Lemma 1

We construct an IND-CCA adversary $\mathcal{A}^\bullet$ that uses adversary $\mathcal{A}^\circ$ to mount an attack against the NewBasicPub^{hy} scheme [12]. The game between the challenger and algorithm $\mathcal{A}^\bullet$ starts with the Initialization stage which we describe below.

Initialization. On input of the security parameter k, the challenger first generates the public key $pk^\star = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, R^\star, Q^\star, m^\star.n, m^\star.n_0, H_1, H_2)$ such that $m^\star \in \{1, \ldots, m_{\vee\wedge}\}$ and $R^\star = s^\star \cdot P$, where $s^\star \in \mathbb{Z}_q^*$ is the private key corresponding to $pk^\star$. Upon receiving $pk^\star$, algorithm $\mathcal{A}^\bullet$ does the following:

1. Let $m^\bullet = m^\star$, then choose the values $m_i^\bullet \in \{1, \ldots, m_\vee\}$ and $m_{i,j}^\bullet \in \{1, \ldots, m_\wedge\}$ $\wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$
2. Pick at random $\alpha_i^\bullet \in \{1, \ldots, m_{\vee\wedge}\}$ and $r_{\alpha_i^\bullet}, \omega_i^\bullet \in \mathbb{Z}_q^*$ $\wr_{i=1}^{m^\bullet}$
3. Pick at random $\beta_{i,j}^\bullet \in \{1, \ldots, m_\vee\}$ and $r_{\beta_{i,j}^\bullet}, \upsilon_{i,j}^\bullet \in \mathbb{Z}_q^*$ $\wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$
4. Pick at random $l_{i,j,k}^\bullet \in \{1, \ldots, q_0\}$, $\gamma_{i,j,k}^\bullet \in \{1, \ldots, m_\wedge\}$ and $r_{\gamma_{i,j,k}^\bullet} \in \mathbb{Z}_q^*$ $\wr_{k=1}^{m_{i,j}^\bullet} \wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$
5. Pick at random $\theta_{i,j,k}^\bullet \in \mathbb{Z}_q^* \wr_{k=2}^{m_{i,j}^\bullet} \wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$, then compute $\theta_{i,j,1}^\bullet = \sum_{k=2}^{m_{i,j}^\bullet} \theta_{i,j,k}^\bullet \wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$
6. Compute $\kappa_{i,j,k}^\bullet = ((\alpha_i^\bullet - 1).m_\vee + \beta_{i,j}^\bullet) - 1).m_\wedge + \gamma_{i,j,k}^\bullet$ and $r_{\kappa_{i,j,k}^\bullet} = r_{\gamma_{i,j,k}^\bullet} r_{\beta_{i,j}^\bullet} r_{\alpha_i^\bullet} \wr_{k=1}^{m_{i,j}^\bullet} \wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$
7. Choose a hash function: $\bar{H}_2^\bullet : \{1, \ldots, m_{\vee\wedge}\} \to \{0,1\}^n$
8. Define the function $\Delta^\bullet : \{0,1\}^{m^\bullet.n} \times \{1 \ldots, m^\bullet\} \to \{0,1\}^n$ which on input of a tuple (X, i) returns the i^{th} block of length n of the binary string X i.e. the bits from $(i-1).n+1$ to $i.n$ of X.
9. Define the functions $\xi_x^f : \{0,1\}^{>x} \to \{0,1\}^x$ and $\xi_x^l : \{0,1\}^{>x} \to \{0,1\}^x$ which on input a string X return, respectively, the first x bits of X and the last x bits of X.
10. Define the function $\Omega^\bullet : \{0,1\}^{m^\bullet.n} \times \{1 \ldots, m^\bullet\} \to \{0,1\}^n$ which on input of a tuple (X, i) returns the binary $\Delta^\bullet(\xi_{m^\bullet.(n-n_0)}^f(X), i) \| \Delta^\bullet(\xi_{m^\bullet.n_0}^l(X), i)$.

Note. We assume that adversary $\mathcal{A}^\circ$ is parameterized with $m^\star \in \mathbb{N}^*$. Furthermore, we assume that $N \geq m_{\vee\wedge} m_\vee$. Our proof can be easily adapted to the case where $N \leq m_{\vee\wedge} m_\vee$. $\diamond$

The interaction between algorithm $\mathcal{A}^\bullet$ and adversary $\mathcal{A}^\circ$ consists of five stages: Setup, Queries-1, Challenge, Queries-2 and Guess which we describe below.

Setup. Algorithm $\mathcal{A}^\bullet$ does the following: (1) Let $I^\bullet = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, n_0, H_0^\bullet, H_1, H_2^\bullet)$ be the global information, where the oracles $H_0^\bullet$ and $H_2^\bullet$ are controlled by algorithm $\mathcal{A}^\bullet$ and the tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, n_0, H_1)$ is taken from *params**, (2) Define the set of trusted authorities $\mathcal{T} = \{TA_1, \ldots, TA_N\}$ as follows: for $\kappa \in \{\kappa_{i,j,k}^\bullet\}$, the public key of TA_κ is $R_\kappa = r_\kappa \cdot R^\star$, whereas, for $\kappa \in \{1, \ldots, N\} \setminus \{\kappa_{i,j,k}^\bullet\}$, the public key of TA_κ is $R_\kappa = s_\kappa \cdot P$ for some randomly chosen $s_\kappa \in \mathbb{Z}_q^*$, (3) Give the global information $I^\bullet$ and the trusted authorities' public keys $R_\kappa \wr_{\kappa=1}^N$ to adversary $\mathcal{A}^\circ$.

Note. For $\kappa \in \{\kappa_{i,j,k}^\bullet\}$, the master key of TA_κ is $s_\kappa = r_\kappa s^\star$

Algorithm $\mathcal{A}^\bullet$ controls the random oracle $H_0^\bullet$ as follows: algorithm $\mathcal{A}^\bullet$ maintains a list of tuples $[A_\iota, H_{0,\iota}, \lambda_\iota]$ which we denote H_0^{list}. The list is initially empty. Assume that adversary $\mathcal{A}^\circ$ makes a query on assertion $A \in \{0,1\}^*$, then adversary $\mathcal{A}^\bullet$ responds as follows:

1. If A already appears on the list H_0^{list} in a tuple $[A_\iota, H_{0,\iota}, \lambda_\iota]$, then return $H_{0,\iota}$
2. If A does not appear on H_0^{list} and A is the $l_{i,j,1}^\bullet$-th distinct query to oracle $H_0^\bullet$, then compute $H_{0,l_{i,j,1}^\bullet} = r_{\gamma_{i,j,1}^\bullet}^{-1} \cdot ((r_{\beta_{i,j}^\bullet}^{-1} \upsilon_{i,j}^\bullet r_{\alpha_i^\bullet}^{-1} \omega_i^\bullet) \cdot Q^\star - \theta_{i,j,1}^\bullet \cdot P)$, return $H_{0,l_{i,j,1}^\bullet}$, and add $[A, H_{0,l_{i,j,1}^\bullet}, \mathsf{null}]$ to H_0^{list}
3. If A does not appear on H_0^{list} and A is the $l_{i,j,k}^\bullet$-th distinct query to oracle $H_0^\bullet$ (for $k > 1$), then compute $H_{0,l_{i,j,k}^\bullet} = (r_{\gamma_{i,j,k}^\bullet}^{-1} \theta_{i,j,k}^\bullet) \cdot P$, return $H_{0,l_{i,j,k}^\bullet}$, and add the entry $[A, H_{0,l_{i,j,k}^\bullet}, r_{\gamma_{i,j,k}^\bullet}^{-1} \theta_{i,j,k}^\bullet]$ to H_0^{list}
4. Otherwise, pick at random $\lambda \in \mathbb{Z}_q^*$ such that $\lambda \cdot P$ does not appear on the list H_0^{list}, return $\lambda \cdot P$, and add the entry $[A, \lambda \cdot P, \lambda]$ to H_0^{list}

Note. The simulated oracle $H_0^\bullet$ is such that $(r_{\beta_{i,j}^\bullet}^{-1}\upsilon_{i,j}^\bullet r_{\alpha_i^\bullet}^{-1}\omega_i^\bullet)\cdot Q^\star = \sum_{k=1}^{m_{i,j}^\bullet} r_{\gamma_{i,j,k}^\bullet} H_{0,l_{i,j,k}^\bullet}\ \wr_{i,j}$

Algorithm $\mathcal{A}^\bullet$ controls the random oracle $H_2^\bullet$ as follows: on input of a tuple (G,i,j), algorithm $\mathcal{A}^\bullet$ returns the value $\Omega^\bullet(H_2(G^{\upsilon_{i,j}^{\bullet\,-1}\omega_i^{\bullet\,-1}})\oplus\bar{H}_2^\bullet(j),i)$.

The policy $Pol_{\text{cr}} = \wedge_{i=1}^{m^\bullet}\vee_{j=1}^{m_i^\bullet}\wedge_{k=1}^{m_{i,j}^\bullet}\langle TA_{\kappa_{i,j,k}^\bullet}, A_{l_{i,j,k}^\bullet}\rangle$ is called the 'crucial' policy. Algorithm $\mathcal{A}^\bullet$ hopes that the 'target' policy Pol_{ch}, which will be chosen by adversary $\mathcal{A}^\circ$ in the Challenge stage of the IND-Pol-CCA game, will be equal to policy Pol_{cr}.

Queries-1. Adversary $\mathcal{A}^\circ$ performs a polynomial number of oracle queries adaptively.

Challenge. Once adversary $\mathcal{A}^\circ$ decides that the stage Queries-1 is over, it outputs two equal length messages M_0 and M_1 as well as a policy Pol_{ch} on which it wishes to be challenged. Algorithm $\mathcal{A}^\bullet$ responds as follows: (1) If $Pol_{\text{ch}} \neq Pol_{\text{cr}}$, then report failure and terminate (we refer to this event as $\mathcal{E}_{\text{ch}}$), (2) Otherwise, first pick at random $M_{d,i} \in \{0,1\}^{n-n_0}\ \wr_{i=1}^{m^{\bullet}-1}$ and set $M_{d,m^\bullet} = M_d \oplus (\oplus_{i=1}^{m^\bullet-1} M_{d,i})\ \wr_{d\in\{0,1\}}$. Then, give the messages $M_d^\bullet = M_{d,1}\|\ldots\|M_{d,m^\bullet}\ \wr_{d\in\{0,1\}}$ to the challenger who picks randomly $b\in\{0,1\}$ and returns a ciphertext $C^\star = (U,v^\star)$ representing the encryption of message $M_b^\bullet$ using the public key $pk^\star$. Upon receiving the challenger's response, compute the values $v_{i,j} = \Omega^\bullet(v^\star\oplus\bar{H}_2^\bullet(j),i)\ \wr_{j=1}^{m_i^\bullet}\wr_{i=1}^{m^\bullet}$, then return the ciphertext $C_{\text{ch}} = (U,[[v_{i,j}]_{j=1}^{m_i^\bullet}]_{i=1}^{m^\bullet})$.

Note. For adversary $\mathcal{A}^\circ$, the ciphertext C_{ch} represents a correct encryption of message M_b according to policy Pol_{ch}. In fact, the ciphertext $C^\star$ is such that $U = H_1(M_b\|t)\cdot P$ (for some randomly chosen $t\in\{0,1\}^{m^\bullet.n}$), and $v^\star = (M_b\|t)\oplus H_2(g^r)$ where $g = e(R^\star,Q^\star)$. Let $t_i = \Omega^\bullet(t,i)$, then the following holds

$$
\begin{aligned}
v_{i,j} &= \Omega^\bullet((M_b\|t)\oplus H_2(e(R^\star,Q^\star)^r)\oplus\bar{H}_2^\bullet(j),i)\\
&= \Omega^\bullet(M_b\|t,i)\oplus\Omega^\bullet(H_2([e((r_{\beta_{i,j}^\bullet}r_{\alpha_i^\bullet})\cdot R^\star,(r_{\beta_{i,j}^\bullet}^{-1}\upsilon_{i,j}^\bullet r_{\alpha_i^\bullet}^{-1}\omega_i^\bullet)\cdot Q^\star)^r]^{\upsilon_{i,j}^{\bullet\,-1}\omega_i^{\bullet\,-1}})\oplus\bar{H}_2^\bullet(j),i)\\
&= (M_i\|t_i)\oplus H_2^\bullet(e((r_{\beta_{i,j}^\bullet}r_{\alpha_i^\bullet})\cdot R^\star,\sum_{k=1}^{m_{i,j}^\bullet} r_{\gamma_{i,j,k}^\bullet}H_{0,l_{i,j,k}^\bullet})^r,i,j) = (M_i\|t_i)\oplus H_2^\bullet([\prod_{k=1}^{m_{i,j}^\bullet} e(R_{\kappa_{i,j,k}^\bullet},H_{0,l_{i,j,k}^\bullet})]^r,i,j)
\end{aligned}
$$

Queries-2. Again, adversary $\mathcal{A}^\circ$ performs a polynomial number of oracle queries adaptively.

Guess. Algorithm $\mathcal{A}^\circ$ outputs a guess b' for b. Algorithm $\mathcal{A}^\bullet$ outputs b' as its guess for b.

The oracles CredGen-O and PolDec-O to which adversary $\mathcal{A}^\circ$ makes queries during Queries-1 and Queries-2 are described below. Without loss of generality, we assume that adversary $\mathcal{A}^\circ$ always makes the appropriate query on A to the random oracle $H_0^\bullet$ before making any query involving A to oracles CredGen-O and PolDec-O.

- **CredGen-O**. Assume that adversary $\mathcal{A}^\circ$ makes a query on a tuple (TA_κ,A). Let $[A_\iota,H_{0,\iota},\lambda_\iota]$ be the tuple from H_0^{list} such that $A_\iota = A$, then algorithm $\mathcal{A}^\bullet$ responds as follows:
  1. If $\iota = l_{i,j,1}^\bullet$ and $\kappa\in\{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, then report failure and terminate (event $\mathcal{E}_{\text{cred}}$)
  2. If $\iota\neq l_{i,j,1}^\bullet$ and $\kappa\in\{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, then return $(r_\kappa\lambda_\iota)\cdot R^\star = (r_\kappa s^\star)\cdot H_{0,\iota}$
  3. If $\kappa\in\{1,\ldots,N\}\setminus\{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, then return $s_\kappa\cdot H_{0,\iota}$
- **PolDec-O**. Assume that adversary $\mathcal{A}^\circ$ makes an oracle query on a tuple $(C,Pol,\{j_1,\ldots,j_m\})$. Then, algorithm $\mathcal{A}^\bullet$ responds as follows:
  1. If $Pol\neq Pol_{\text{ch}}$ and Pol involves a condition $\langle TA_\kappa,A\rangle$ such that $\kappa\in\{\kappa_{i,j,k}^\bullet\}$ and $A\in\{A_{l_{i,j,1}^\bullet}\}$, then report failure and terminate (event $\mathcal{E}_{\text{dec}}$)
  2. If $Pol\neq Pol_{\text{ch}}$ and Pol does not involve any condition $\langle TA_\kappa,A\rangle$ such that $\kappa\in\{\kappa_{i,j,k}^\bullet\}$ and $A\in\{A_{l_{i,j,1}^\bullet}\}$, then do the following: (1) Run oracle CredGen-O multiple times until obtaining the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, (2) Run algorithm PolDec on input the tuple $(C,Pol,\varsigma_{j_1,\ldots,j_m}(Pol))$ and return the resulting output back to adversary $\mathcal{A}^\circ$

3. If $Pol = Pol_{\mathsf{ch}}$, then do the following: let $C = (U, [[v_{i,j}]_{j=1}^{m_i^\bullet}]_{i=1}^{m^\bullet})$, then compute the values $v_i^\bullet = v_{i,j_i} \oplus \bar{H}_2^\bullet(j_i)\zeta_{i=1}^{m^\bullet}$, and make a decryption query to the challenger on ciphertext $C^\bullet = (U, \xi_{n-n_0}^f(v_1^\bullet)\| \ldots \|\xi_{n-n_0}^f(v_{m^\bullet}^\bullet)\|\xi_{n_0}^l(v_1^\bullet)\| \ldots \|\xi_{n_0}^l(v_{m^\bullet}^\bullet))$. Upon receiving the challenger's response, forward it to adversary $\mathcal{A}^\circ$

In the following, we analyze the simulation described above:

If algorithm $\mathcal{A}^\bullet$ does not report failure during the simulation, then the view of algorithm $\mathcal{A}^\circ$ is identical to its view in the real attack. In fact, observe first that the responses of algorithm $\mathcal{A}^\bullet$ to all queries of adversary $\mathcal{A}^\circ$ to oracle $H_0^\bullet$ are uniformly and independently distributed in group $\mathbb{G}_1$ as in the real IND-Pol-CCA attack. Second, all the responses of algorithm $\mathcal{A}^\bullet$ to queries made by adversary $\mathcal{A}^\circ$ to oracles CredGen-O and PolSig-O are consistent. Third, the ciphertext C_{ch} given to adversary $\mathcal{A}^\circ$ corresponds to the encryption according to Pol_{ch} of M_b for some random $b \in \{0,1\}$.

Algorithm $\mathcal{A}^\bullet$ reports failure if either event $\mathcal{E}_{\mathsf{ch}}$, event $\mathcal{E}_{\mathsf{cred}}$ or event $\mathcal{E}_{\mathsf{dec}}$ occurs during the simulation. Since events $\mathcal{E}_{\mathsf{cred}}$ and $\mathcal{E}_{\mathsf{dec}}$ are independent, the following statement holds

$$\mathrm{Adv}_{\mathcal{A}^\bullet} \geq Pr[\neg\mathcal{E}_{\mathsf{cred}} \wedge \neg\mathcal{E}_{\mathsf{ch}} \wedge \neg\mathcal{E}_{\mathsf{dec}}].\varepsilon \geq Pr[\neg\mathcal{E}_{\mathsf{ch}}|\neg\mathcal{E}_{\mathsf{cred}} \wedge \neg\mathcal{E}_{\mathsf{dec}}].Pr[\neg\mathcal{E}_{\mathsf{cred}}].Pr[\neg\mathcal{E}_{\mathsf{dec}}].\varepsilon \quad (1)$$

From the simulation described above, we have

$$Pr[\mathcal{E}_{\mathsf{cred}}] \leq \frac{q_c m_{\vee\wedge} m_{\vee}}{N q_0} \quad (2)$$

Adversary $\mathcal{A}^\bullet$ picks the challenge policy from a set of $\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ distinct policies. Then, the following statement holds

$$Pr[\neg\mathcal{E}_{\mathsf{ch}}|\neg\mathcal{E}_{\mathsf{cred}} \wedge \neg\mathcal{E}_{\mathsf{dec}}] \geq \frac{1}{\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})} \quad (3)$$

The total number of policies, distinct from policy Pol_{ch}, that may be specified by adversary $\mathcal{A}^\bullet$ during queries to oracle PolDec-O, and that involve at least one of the conditions $\langle TA_\kappa, A\rangle$ such that $\kappa \in \{\kappa_{i,j,k}^\bullet\}$ and $A \in \{A_{l_{i,j,1}^\bullet}\}$ could be upper bounded by $\Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = \Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) - \Upsilon(Nq_0 - (m_{\vee\wedge}m_{\vee})^2, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) - 1$. Then, the following statement holds

$$Pr[\mathcal{E}_{\mathsf{dec}}] \leq \frac{q_d \Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})}{\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})} \quad (4)$$

Finally, statements (1), (2), (3) and (4) lead to the result

$$F(q_c, q_d, q_0, N, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = (1 - \frac{q_c m_{\vee\wedge} m_{\vee}}{N q_0}).(1 - \frac{q_d \Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})}{\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})}).\frac{1}{\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge})}$$

Note. In the particular case where $N = m_{\vee\wedge} = m_{\vee} = m_{\wedge} = 1$, our PBE scheme is equivalent to the New-FullIdent scheme of [12]. In this case, note that our results match Result 5 of [12]. In fact, in this case we have $\Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = 0$ and $\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = q_0$. $\diamond$

B Proof of Lemma 2

We construct an algorithm $\mathcal{A}^\bullet$ that uses $\mathcal{A}^\circ$ to mount an attack against CDHP. The game between the challenger and algorithm $\mathcal{A}^\bullet$ starts with the Initialization stage which we describe below.

Initialization. The challenger gives to adversary $\mathcal{A}^\bullet$ the BDH parameters $(q, \mathbb{G}_1, \mathbb{G}_2, e, P)$ as well as a CDHP-instance $(P, a \cdot P, b \cdot P) = (P, P_1, P_2)$ for these parameters. Then, algorithm $\mathcal{A}^\bullet$ does the following:

1. Choose the values $i^\bullet \in \{1, \ldots, m_{\vee\wedge}\}$, $j^\bullet \in \{1, \ldots, m_\vee\}$ and $m^\bullet_{i^\bullet,j^\bullet} \in \{1, \ldots, m_\wedge\}$
2. Pick at random the values $\kappa^\bullet_{i^\bullet,j^\bullet,k} \in \{1, \ldots, N\}$ and $l_{i^\bullet,j^\bullet,k} \in \{1, \ldots, q_0\}$ $\wr_{k=1}^{m^\bullet_{i^\bullet,j^\bullet}}$
3. Pick at random $\theta^\bullet_{i^\bullet,j^\bullet,k} \in \mathbb{Z}_q^*$ $\wr_{k=2}^{m^\bullet_{i^\bullet,j^\bullet}}$, then compute $\theta^\bullet_{i^\bullet,j^\bullet,1} = \sum_{k=2}^{m^\bullet_{i^\bullet,j^\bullet}} \theta^\bullet_{i^\bullet,j^\bullet,k}$

The interaction between algorithm $\mathcal{A}^\bullet$ and adversary $\mathcal{A}^\circ$ consists of three stages: Setup, Probing and Forge which we describe below.

Setup. Algorithm $\mathcal{A}^\bullet$ does the following: (1) Let $I^\bullet = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_0^\bullet, H_4^\bullet)$ be the global information, where the oracles $H_0^\bullet$ and $H_4^\bullet$ are controlled by algorithm $\mathcal{A}^\bullet$, the tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P)$ is given to algorithm $\mathcal{A}^\bullet$ in the Initialization stage, and the value $n \in \mathbb{N}^*$ is chosen by algorithm $\mathcal{A}^\bullet$, (2) Define the set of trusted authorities $\mathcal{T} = \{TA_1, \ldots, TA_N\}$ as follows: for $\kappa \in \{\kappa^\bullet_{i^\bullet,j^\bullet,k}\}$, the public key of TA_κ is $R_\kappa = r_\kappa \cdot P_1$ for some randomly chosen $r_\kappa \in \mathbb{Z}_q^*$, whereas, for $\kappa \in \{1, \ldots, N\} \setminus \{\kappa^\bullet_{i^\bullet,j^\bullet,k}\}$, the public key of TA_κ is $R_\kappa = s_\kappa \cdot P$ for some randomly chosen $s_\kappa \in \mathbb{Z}_q^*$, (3) Give the global information $I^\bullet$ and the trusted authorities' public keys R_κ $\wr_{\kappa=1}^{N}$ to adversary $\mathcal{A}^\circ$.

Note. For $\kappa \in \{\kappa^\bullet_{i^\bullet,j^\bullet,k}\}$, the master key of TA_κ is $s_\kappa = r_\kappa a$

Algorithm $\mathcal{A}^\bullet$ controls the random oracle $H_0^\bullet$ as follows: algorithm $\mathcal{A}^\bullet$ maintains a list of tuples $[A_\iota, H_{0,\iota}, \lambda_\iota]$ which we denote H_0^{list}. The list is initially empty. Assume that adversary $\mathcal{A}^\circ$ makes a query on assertion $A \in \{0,1\}^*$, then adversary $\mathcal{A}^\bullet$ responds as follows:

1. If A already appears on the list H_0^{list} in a tuple $[A_\iota, H_{0,\iota}, \lambda_\iota]$, then return $H_{0,\iota}$
2. If A does not appear on H_0^{list} and A is the $l^\bullet_{i^\bullet,j^\bullet,1}$-th distinct query to oracle $H_0^\bullet$, then compute $H_{0,l^\bullet_{i,j,1}} = r^{-1}_{\kappa^\bullet_{i^\bullet,j^\bullet,1}} \cdot (P_2 - \theta^\bullet_{i^\bullet,j^\bullet,1} \cdot P)$, return $H_{0,l^\bullet_{i^\bullet,j^\bullet,1}}$, and add the entry $[A, H_{0,l^\bullet_{i^\bullet,j^\bullet,1}}, \mathsf{null}]$ to H_0^{list}
3. If A does not appear on H_0^{list} and A is the $l^\bullet_{i^\bullet,j^\bullet,k}$-th distinct query to oracle $H_0^\bullet$ (for $k > 1$), then compute $H_{0,l^\bullet_{i^\bullet,j^\bullet,k}} = (r^{-1}_{\kappa^\bullet_{i^\bullet,j^\bullet,k}} \theta^\bullet_{i^\bullet,j^\bullet,k}) \cdot P$, return $H_{0,l^\bullet_{i^\bullet,j^\bullet,k}}$, and add $[A, H_{0,l^\bullet_{i^\bullet,j^\bullet,k}}, r^{-1}_{\kappa^\bullet_{i^\bullet,j^\bullet,k}} \theta^\bullet_{i^\bullet,j^\bullet,k}]$ to H_0^{list}
4. Otherwise, pick at random $\lambda \in \mathbb{Z}_q^*$, return $\lambda \cdot P$ and add $[A, \lambda \cdot P, \lambda]$ to H_0^{list}

Note. The random oracle $H_0^\bullet$ is such that the following holds

$$\begin{aligned}
\tau^\bullet_{i^\bullet,j^\bullet} &= \prod_{k=1}^{m^\bullet_{i^\bullet,j^\bullet}} e(R_{\kappa_{i^\bullet,j^\bullet,k}}, H_0(A_{i^\bullet,j^\bullet,k})) \\
&= e(r_{\kappa_{i^\bullet,j^\bullet,1}} \cdot P_1, r^{-1}_{\kappa_{i^\bullet,j^\bullet,1}} \cdot (P_2 - \theta_{i^\bullet,j^\bullet,1} \cdot P)) * \prod_{k=2}^{m^\bullet_{i^\bullet,j^\bullet}} e(r_{\kappa_{i^\bullet,j^\bullet,k}} \cdot P_1, (r^{-1}_{\kappa_{i^\bullet,j^\bullet,k}} \theta_{i^\bullet,j^\bullet,k}) \cdot P) \\
&= e(P_1, P_2 - (\theta_{i^\bullet,j^\bullet,1} - \sum_{k=2}^{m^\bullet_{i^\bullet,j^\bullet}} \theta_{i^\bullet,j^\bullet,k}) \cdot P) = e(P_1, P_2) = e(P, ab \cdot P) \qquad \diamond
\end{aligned}$$

Probing. Adversary $\mathcal{A}^\circ$ performs a polynomial number of oracle queries adaptively.

Forging. Algorithm $\mathcal{A}^\circ$ outputs a message M_f, a policy Pol_f and a signature σ_f. The adversary wins the game if $\mathsf{PolVrf}(M_\mathrm{f}, Pol_\mathrm{f}, \sigma_\mathrm{f}) = \top$.

The oracles that adversary $\mathcal{A}^{\circ}$ may query during Probing are defined below. We assume without loss of generality that adversary $\mathcal{A}^{\circ}$ always makes the appropriate query to the random oracle $H_0^{\bullet}$ on assertion A.

- **CredGen-O**. Assume that adversary $\mathcal{A}^{\circ}$ makes a query on a tuple (TA_{κ}, A). Let $[A_{\iota}, H_{0,\iota}, \lambda_{\iota}]$ be the tuple from H_0^{list} such that $A_{\iota} = A$, then algorithm $\mathcal{A}^{\bullet}$ responds as follows:
  1. If $\iota = l^{\bullet}_{i^{\bullet},j^{\bullet},1}$ and $\kappa \in \{\kappa^{\bullet}_{i^{\bullet},j^{\bullet},k}\}$, then report failure and terminate (event $\mathcal{E}_{\mathsf{cred}}$)
  2. If $\iota \neq l^{\bullet}_{i^{\bullet},j^{\bullet},1}$ and $\kappa \in \{\kappa^{\bullet}_{i^{\bullet},j^{\bullet},k}\}$, then return $(r_{\kappa}\lambda_{\iota}) \cdot P_1 = (r_{\kappa}a) \cdot H_{0,\iota} = s_{\kappa} \cdot H_{0,\iota}$
  3. If $\kappa \in \{1, \ldots, N\} \setminus \{\kappa^{\bullet}_{i^{\bullet},j^{\bullet},k}\}$, then return $s_{\kappa} \cdot H_{0,\iota}$
- **PolSig-O**. Assume that adversary $\mathcal{A}^{\circ}$ makes a query on a tuple (M, Pol). Algorithm $\mathcal{A}^{\bullet}$ responds as follows:
  1. Pick at random $h_{i,1} \in \mathbb{Z}_q^*$ $\wr_{i=1}^{m}$ and $Y_{i,j} \in \mathbb{G}_1$ $\wr_{j=1}^{m_i} \wr_{i=1}^{m}$
  2. Compute $\tau_{i,j} = \prod_{k=1}^{m_{i,j}} e(R_{\kappa_{i,j,k}}, H_0^{\bullet}(A_{i,j,k}))$ $\wr_{j=1}^{m_i} \wr_{i=1}^{m}$
  3. Compute $x_{i,j+1} = e(P, Y_{i,j}) * \tau_{i,j}^{h_{i,j}}$, then compute $h_{i,j+1} = H_4(M \| x_{i,j+1} \| m \| i \| j+1)$ $\wr_{j=1}^{m_i-1} \wr_{i=1}^{m}$. In order to compute the value $h_{i,j}$, algorithm $\mathcal{A}^{\bullet}$ maintains a list of tuples $[(M_{\iota}, x_{\iota}, m_{\iota}, i_{\iota}, j_{\iota}), H_{4,\iota}]$ which we denote H_4^{list}. If $(M, x_{i,j}, m, i, j)$ appears on H_4^{list} in a tuple $[(M_{\iota}, x_{\iota}, m_{\iota}, i_{\iota}, j_{\iota}), H_{4,\iota}]$, then algorithm $\mathcal{A}^{\bullet}$ sets $h_{i,j} = H_{4,\iota}$. Otherwise, it picks at random $H \in \mathbb{Z}_q^*$, sets $h_{i,j} = H$ and adds the tuple $[(M, x_{i,j}, m, i, j), H]$ to H_4^{list}.
  4. Let $x_{i,1} = e(P, Y_{i,m_i}) * \tau_{i,m_i}^{h_{i,m_i}}$ and $h_{i,1} = H_4(M \| x_{i,1} \| m \| i \| 1)$, then
     (a) If $(M, x_{i,1}, m, i, 1)$ already appears on the list H_4^{list} in a tuple $[(M_{\iota}, x_{\iota}, m_{\iota}, i_{\iota}, 1), H_{4,\iota}]$ such that $H_{4,\iota} \neq h_{i,1}$, then report failure and terminate (we refer to this event as $\mathcal{E}_{\mathsf{sig}}$).
     (b) Otherwise, add the tuple $[(M, x_{i,1}, m, i, 1), h_{i,1}]$ to H_4^{list}.
  5. Compute $Y = \sum_{i=1}^{m} \sum_{j=1}^{m_i} Y_{i,j}$, then return $([x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y)$ to adversary $\mathcal{A}^{\circ}$.

Algorithm $\mathcal{A}^{\bullet}$ controls the random oracle $H_4^{\bullet}$ as follows: assume that adversary $\mathcal{A}^{\circ}$ makes a query to the random oracle $H_4^{\bullet}$ on input (M, x, m, i, j), then algorithm $\mathcal{A}^{\bullet}$ responds as follows:

1. If the tuple (M, x, m, i, j) already appears on H_4^{list} in a tuple $[(M_{\iota}, x_{\iota}, m_{\iota}, i_{\iota}, j_{\iota}), H_{4,\iota}]$, then output $H_{4,\iota}$
2. Otherwise, pick at random $H \in \mathbb{Z}_q^*$, output H and add $[(M, x, m, i, j), H]$ to H_4^{list}

In the following, we analyze the simulation described above:

Let w be the whole set of random tapes that take part in an attack by adversary $\mathcal{A}^{\circ}$, with the environment simulated by algorithm $\mathcal{A}^{\bullet}$, but excluding the randomness related to the oracle $H_4^{\bullet}$. The success probability of adversary $\mathcal{A}^{\circ}$ in forging a valid ring signature scheme is then taken over the space $(w, H_4^{\bullet})$. Let $\mathcal{S}$ be the set of successful executions of adversary $\mathcal{A}^{\circ}$, then the following holds

$$\mathrm{Adv}_{\mathcal{A}^{\circ}} = Pr[(w, H_4^{\bullet}) \in \mathcal{S}] \geq \varepsilon \tag{5}$$

Let $\mathcal{E}_0$ be the event that adversary $\mathcal{A}^{\circ}$ succeeds in forging the signature $\sigma_{\mathrm{f}} = ([x_{i,j}^{\mathrm{f}}]_{j=1}^{m_i}]_{i=1}^{m}, Y^{\mathrm{f}})$ without making a query to the random oracle $H_4^{\bullet}$ on at least one of the tuples $(M_{\mathrm{f}}, x_{i,j}^{\mathrm{f}}, m, i, j)$. Then, the following holds

$$Pr[\mathcal{E}_0] \leq \frac{m_{\vee\wedge} m_{\vee}}{q} \tag{6}$$

Let $\mathcal{E}_0'$ be the event that event $\mathcal{E}_{\mathsf{sig}}$ occurs at one of the queries made by adversary $\mathcal{A}^{\circ}$ to the oracle PolSig-O. Then, the following holds

$$Pr[\mathcal{E}_0'] \leq \frac{m_{\vee\wedge} q_s q_4}{q} \tag{7}$$

Let $\mathcal{S}'$ be the set of successful executions of adversary $\mathcal{A}^{\circ}$ for which it has made queries to the random oracle $H_4^{\bullet}$ on the all the tuples $(M_{\mathrm{f}}, x_{i,j}^{\mathrm{f}}, m, i, j)$, then the following holds

$$\begin{aligned} Pr[(w, H_4^{\bullet}) \in \mathcal{S}'] &= Pr[\neg \mathcal{E}_0].Pr[\neg \mathcal{E}_0'].Pr[(w, H_4^{\bullet}) \in \mathcal{S}] \\ &\geq (1 - \frac{m_{\vee\wedge} m_{\vee}}{q}).(1 - \frac{m_{\vee\wedge} q_s q_4}{q}).\varepsilon \end{aligned} \tag{8}$$

Let $Q_1, \ldots, Q_{q_4}$ denote the different queries made by adversary $\mathcal{A}^{\circ}$ to the random oracle $H_4^{\bullet}$. We denote by $Q_{\beta_{i,j}}$ (for $\beta_{i,j} \in \{1, \ldots, q_4\}$) the query made by adversary $\mathcal{A}^{\circ}$ to the random oracle $H_4^{\bullet}$ on the tuple $(M_{\mathrm{f}}, x_{i,j}^{\mathrm{f}}, m, i, j)$. Let i^{lq} and j^{lq} be the indexes such that for all $(i, j) \neq (i^{lq}, j^{lq})$, $\beta_{i,j} < \beta_{i^{lq}, j^{lq}}$. The value $\beta_{i^{lq}, j^{lq}}$ is called the last-query index. We define $\mathcal{S}'_{\beta_{i^{lq}, j^{lq}}}$ to be the set of executions from $\mathcal{S}'$ whose last-query index is $\beta_{i^{lq}, j^{lq}}$. Since $\beta_{i^{lq}, j^{lq}}$ may range between $\beta \leq \beta = m_{\vee\wedge} m_{\vee}$ and q_4, this gives us a partition of $\mathcal{S}'$ in at least $q_4 + 1 - \beta$ classes.

Let $\mathcal{E}_1$ be the event that algorithm $\mathcal{A}^{\bullet}$ obtains a successful execution $(w^1, H_4^{\bullet 1}) \in \mathcal{S}'_{\beta^1_{i^{lq}, j^{lq}}}$, for some last-query index $\beta^1_{i^{lq}, j^{lq}}$, after invoking t_1 times adversary $\mathcal{A}^{\circ}$ with randomly chosen tuples $(w, H_4^{\bullet})$. In the particular case where $t_1 = (Pr[w, H_4^{\bullet}) \in \mathcal{S}'])^{-1}$, and since $(1 - \frac{1}{X})^X \leq e^{-1}$ (for $X > 1$), the following statement holds

$$Pr[\mathcal{E}_1] = 1 - (1 - Pr[(w, H_4^{\bullet}) \in \mathcal{S}'])^{t_1} \geq 1 - e^{-1} > \frac{3}{5} \tag{9}$$

We define the set $\mathcal{J}$ of last-query indexes which are more likely to appear as follows:

$$\mathcal{J} = \{\beta_{i^{lq}, j^{lq}} \; s.t. \; Pr[(w, H_4^{\bullet}) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}} | (w, H_4^{\bullet}) \in \mathcal{S}'] \geq \gamma\}, \text{ where } \gamma = \tfrac{1}{2(q_4 + 1 - \beta)}$$

Let $\mathcal{S}'_{\mathcal{J}} = \{(w, H_4^{\bullet}) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}} \; s.t. \; \beta_{i^{lq}, j^{lq}} \in \mathcal{J}\}$ be the subset of successful executions corresponding to the set $\mathcal{J}$. Since the subsets $\mathcal{S}'_{\beta_{i^{lq}, j^{lq}}}$ are pairwise disjoint, the following holds

$$\begin{aligned} Pr[(w, H_4^{\bullet}) \in \mathcal{S}'_{\mathcal{J}} | (w, H_4^{\bullet}) \in \mathcal{S}'] &= \sum_{\beta_{i^{lq}, j^{lq}} \in \mathcal{J}} Pr[(w, H_4^{\bullet}) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}} | (w, H_4^{\bullet}) \in \mathcal{S}'] \\ &= 1 - \sum_{\beta_{i^{lq}, j^{lq}} \notin \mathcal{J}} Pr[(w, H_4^{\bullet}) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}} | (w, H_4^{\bullet}) \in \mathcal{S}'] \\ &\geq 1 - (\frac{1}{2\gamma} - |\mathcal{J}|).\gamma \geq \frac{1}{2} \end{aligned} \tag{10}$$

Let $\alpha = (1 - \frac{m_{\vee\wedge} m_{\vee}}{q}).(1 - \frac{m_{\vee\wedge} q_s q_4}{q}).\varepsilon.\gamma$, then equation (8) leads to the following statement

$$Pr[(w, H_4^{\bullet}) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}}] = Pr[(w, H_4^{\bullet}) \in \mathcal{S}'].Pr[(w, H_4^{\bullet}) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}} | (w, H_4^{\bullet}) \in \mathcal{S}'] \geq \alpha \tag{11}$$

The oracle $H_4^{\bullet}$ can be written as a pair $(\tilde{H}_4, h_{i^{lq}, j^{lq}})$, where $\tilde{H}_4$ corresponds to the answers for all the queries to oracle $H_4^{\bullet}$ except the query $Q_{\beta_{i^{lq}, j^{lq}}}$ whose answer is denoted as $h_{i^{lq}, j^{lq}}$. We define the set $\Omega_{\beta_{i^{lq}, j^{lq}}}$ as follows:

$$\Omega_{\beta_{i^{lq}, j^{lq}}} = \{(w, (\tilde{H}_4, h_{i^{lq}, j^{lq}})) \in \mathcal{S}' \; s.t. \; Pr_{h_{i^{lq}, j^{lq}}}[(w, (\tilde{H}_4, h_{i^{lq}, j^{lq}})) \in \mathcal{S}'_{\beta_{i^{lq}, j^{lq}}}] \geq \delta - \alpha\}$$

Lemma 3. (Splitting Lemma defined in [20]) *Let $A \subset X \times Y$ s.t. $Pr[(x, y) \in A] \geq \varepsilon$. Given $\alpha < \varepsilon$, define the set $B = \{(x, y) \in X \times Y | Pr_{y' \in Y}[(x, y') \in A] \geq \varepsilon - \alpha\}$. The following statements hold:*

1. $Pr[(x, y) \in B | (x, y) \in X \times Y] \geq \alpha$

2. $\forall\ (x,y) \in B,\ Pr_{y' \in Y}[(x,y') \in A] \geq \varepsilon - \alpha$
3. $Pr[(x,y) \in B | (x,y) \in A] \geq \frac{\alpha}{\varepsilon}$

For $\delta = 2\alpha$ and according to the splitting lemma, the following statements hold

$$\forall\ (w,(\tilde{H}_4, h_{i^{l_q},j^{l_q}})) \in \Omega_{\beta_{i^{l_q},j^{l_q}}},\ Pr_{h_{i^{l_q},j^{l_q}}}[(w,(\tilde{H}_4, h_{i^{l_q},j^{l_q}})) \in \mathcal{S}'_{\beta_{i^{l_q},j^{l_q}}}] \geq \alpha \tag{12}$$

$$Pr[(w,(\tilde{H}_4, h_{i^{l_q},j^{l_q}})) \in \Omega_{\beta_{i^{l_q},j^{l_q}}} | (w,(\tilde{H}_4, h_{i^{l_q},j^{l_q}})) \in \mathcal{S}'_{\beta_{i^{l_q},j^{l_q}}}] \geq \frac{\alpha}{\delta} = \frac{1}{2} \tag{13}$$

Assume that event $\mathcal{E}_1$ occurs and that the successful execution $(w^1,(\tilde{H}_4^1, h^1_{i^{l_q},j^{l_q}}))$ is in $\mathcal{S}'_g$. Let $\mathcal{E}_2$ be the event that algorithm $\mathcal{A}^\bullet$ obtains, for some last-query index $\beta^1_{i^{l_q},j^{l_q}}$, a successful execution $(w^1,(\tilde{H}_4^1, h^2_{i^{l_q},j^{l_q}})) \in \Omega_{\beta^1_{i^{l_q},j^{l_q}}}$ such that $h^2_{i^{l_q},j^{l_q}} \neq h^1_{i^{l_q},j^{l_q}}$, after invoking t_2 times adversary $\mathcal{A}^\circ$, with fixed $(w^1,\tilde{H}_4^1)$ and randomly chosen $h_{i^{l_q},j^{l_q}}$. In the particular case where $t_2 = (\alpha - \frac{1}{q})^{-1}$, the following holds

$$Pr[\mathcal{E}_2] = 1 - (1 - (\alpha - \frac{1}{q}))^{t_2} \geq 1 - e^{-1} > \frac{3}{5} \tag{14}$$

Consider $(w^1,(\tilde{H}_4^1, h^1_{i^{l_q},j^{l_q}}))$ and $(w^1,(\tilde{H}_4^1, h^2_{i^{l_q},j^{l_q}}))$, the two successful executions of the attack obtained by algorithm $\mathcal{A}^\bullet$ if events $\mathcal{E}_1$ and $\mathcal{E}_2$ occur. For the two considered executions, the random tapes w are identical, whereas the answers of the random oracle $H_4^\bullet$ to the queries of adversary $\mathcal{A}^\circ$ are identical only until the query $Q_{\beta^1_{i^{l_q},j^{l_q}}}$.

Let $\sigma_{\mathrm{f}}^1 = ([x^1_{i,j}]_{j=1}^{m_i^1}]_{i=1}^{m^1}, Y^1)$ and $\sigma_{\mathrm{f}}^2 = ([x^2_{i,j}]_{j=1}^{m_i^2}]_{i=1}^{m^2}, Y^2)$ be the signatures forged by adversary $\mathcal{A}^\circ$ through the two considered successful executions respectively. With probability greater than $\frac{1}{\sum_{l=1}^{m_\vee} l!\binom{m_\vee}{l}}$, we have $m^1 = m^2 = m$ and $m_i^1 = m_i^2 = m_i\ \wr_{i=1}^{m}$. In this case, the following statements hold

1. $x^1_{i,j} = x^2_{i,j}\ \wr_{j=1}^{m_i} \wr_{i=1}^{m}$
2. $h^1_{i,j} = h^2_{i,j}\ \wr_{j \neq j^{l_q}} \wr_{i \neq i^{l_q}}\ ,\ h^1_{i^{l_q},j^{l_q}} \neq h^2_{i^{l_q},j^{l_q}}$

The fact that σ_{f}^1 and σ_{f}^2 are valid ring signatures leads to the equality $e(P, Y^2 - Y^1) = \tau_{i^{l_q},j^{l_q}}^{h^1_{i^{l_q},j^{l_q}} - h^2_{i^{l_q},j^{l_q}}}$. With probability greater than $1/q_0^{m_{\vee\wedge}m_\vee}$, we have $\tau_{i^\bullet,j^\bullet} = \tau_{i^{l_q},j^{l_q}}$. In this case, note that adversary $\mathcal{A}^\circ$ does not make a query to oracle $\mathsf{CredGen}$ on assertion $A_{l^\bullet_{i^\bullet,j^\bullet},1}$ (event $\mathcal{E}_{\mathsf{cred}}$ does not occur). This case leads to the equality $\tau_{i^{l_q},j^{l_q}} = e(P, ab \cdot P)$, and so $Y^2 - Y^1 = (h^1_{i^{l_q},j^{l_q}} - h^2_{i^{l_q},j^{l_q}}) \cdot (ab \cdot P)$. Thus, with probability $Pr[\mathcal{A}^\bullet \text{ wins}]$, algorithm $\mathcal{A}^\bullet$ succeeds to obtain $ab \cdot P$ by computing the quantity $(h^1_{i^{l_q},j^{l_q}} - h^2_{i^{l_q},j^{l_q}})^{-1} \cdot (Y^2 - Y^1)$. From statements (9), (10), (13) and (14), we have $\mathrm{Adv}_{\mathcal{A}^\bullet} = Pr[\mathcal{A}^\bullet \text{ wins}] \geq \frac{9}{100 q_0^{m_{\vee\wedge}m_\vee}} \cdot \frac{1}{\sum_{l=1}^{m_\vee} l!\binom{m_\vee}{l}}$.

For $q \geq Max\{2m_{\vee\wedge}m_\vee, 2m_{\vee\wedge}q_s q_4\}$ and $\varepsilon \leq 32(q_4 + 1 - m_{\vee\wedge}m_\vee)/q$, the running time of adversary $\mathcal{A}^\bullet$ is such that the following holds

$$t_{\mathcal{A}^\bullet} = (t_1 + t_2).t_{\mathcal{A}^\circ} = (\frac{\gamma}{\alpha} + \frac{1}{\alpha - 1/q}).t_{\mathcal{A}^\circ} \leq (\frac{4}{\varepsilon} + \frac{32(q_4 + 1 - m_{\vee\wedge}m_\vee)}{\varepsilon}).t_{\mathcal{A}^\circ} \leq \frac{32q_4 + 4}{\varepsilon}.t_{\mathcal{A}^\circ}$$

C A PBE scheme from the Boneh-Franklin IBE scheme

In this section, we present a PBE scheme (denoted BF-PBE) which may be seen as an extension to the policy setting of the original Boneh-Franklin IBE scheme [8]. After describing the scheme, we analyze its security under the IND-Pol-CCA model.

C.1 Description

The BF-PBE scheme consists of the algorithms described below.

Setup. On input of a security parameter k, do the following: (1) Run algorithm BDH-Setup to obtain a tuple $(q,\mathbb{G}_1,\mathbb{G}_2,e,P)$, (2) Let $\mathcal{M}=\{0,1\}^n$ and $\mathcal{C}=\mathbb{G}_1\times(\{0,1\}^n)^*\times\{0,1\}^n$ (for some $n\in\mathbb{N}^*$), (3) Define four hash functions: $H_0:\{0,1\}^*\rightarrow\mathbb{G}_1$, $H_1:\{0,1\}^*\rightarrow\mathbb{Z}_q^*$, $H_2:\{0,1\}^*\rightarrow\{0,1\}^n$ and $H_3:\{0,1\}^*\rightarrow\{0,1\}^n$, Let $I=(q,\mathbb{G}_1,\mathbb{G}_2,e,P,n,H_0,H_1,H_2,H_3)$.

TA-Setup. Let $\mathcal{T}=\{TA_1,\ldots,TA_N\}$ be a set of trusted authorities. Each trusted authority $TA_\kappa\in\mathcal{T}$ picks at random a secret master key $s_\kappa\in\mathbb{Z}_q^*$ and publishes the corresponding public key $R=s_\kappa\cdot P$.

CredGen. On input of $TA_\kappa\in\mathcal{T}$ and $A\in\{0,1\}^*$, this algorithm outputs $\varsigma(R_\kappa,A)=s_\kappa\cdot H_0(A)$.

PolEnc. On input of message $M\in\mathcal{M}$ and policy Pol, do the following:

1. Pick at random $t_i\in\{0,1\}^n$ $\wr_{i=1}^m$
2. Compute $r=H_1(M\|t_1\|\ldots\|t_a)$, then compute $U=r\cdot P$
3. Compute $\pi_{i,j}=\prod_{k=1}^{m_{i,j}}e(R_{\kappa_{i,j,k}},H_0(A_{i,j,k}))$ $\wr_{j=1}^{m_i}\wr_{i=1}^m$
4. Compute $\mu_{i,j}=H_2(\pi_{i,j}^r\|i\|j)$, then compute $v_{i,j}=t_i\oplus\mu_{i,j}$ $\wr_{j=1}^{m_i}\wr_{i=1}^m$
5. Compute $W=M\oplus H_3(t_1\|\ldots\|t_a)$
6. Return $C=(U,[[v_{i,j}]_{j=1}^{m_i}]_{i=1}^m,W)$

PolDec. On input of ciphertext $C=(U,[[v_{i,j}]_{j=1}^{m_i}]_{i=1}^m,W)$, policy Pol, and the qualified set of credentials $\varsigma_{j_1,\ldots,j_a}(Pol)$, do the following:

1. Compute $\tilde{\pi}_{i,j_i}=e(U,\sum_{k=1}^{m_{i,j_i}}\varsigma(R_{\kappa_{i,j_i,k}},A_{i,j_i,k}))$ $\wr_{i=1}^m$
2. Compute $\tilde{\mu}_{i,j_i}=H_2(\tilde{\pi}_{i,j_i}\|i\|j_i)$, then compute $t_i=v_{i,j_i}\oplus\tilde{\mu}_{i,j_i}$ $\wr_{i=1}^m$
3. Compute $M=W\oplus H_3(t_1\|\ldots\|t_a)$, then compute $r=H_1(M\|t_1\|\ldots\|t_a)$
4. If $U=r\cdot P$, then return the message M, otherwise return $\perp$

Note. The BF-PBE scheme is such that the decryption information $\varphi_{j_1,\ldots,j_m}(C=(U,[[v_{i,j}]_{j=1}^{m_i}]_{i=1}^m,W),Pol)$ consists of the values U,W and the pairs $(v_{i,j_i},\wedge_{k=1}^{m_{i,j_i}}\langle TA_{\kappa_{i,j_i,k}},A_{i,j_i,k}\rangle)$ $\wr_{i=1}^m$. $\diamond$

C.2 Security

Theorem 4. *The* BF-PBE *scheme is* IND-Pol-CCA *secure in the random oracle model under the assumption that BDHP is hard.*

Proof. Theorem 4 follows from a sequence of three reduction arguments: (1) In [8], the public-key encryption scheme BasicPub is shown to be IND-CPA secure in the random oracle model under the assumption that BDHP is hard, (2) In [12], it is shown that an IND-CCA attack on BasicPub^{hy} can be converted into an IND-CPA attack on BasicPub, (3) Lemma 4 shows that an IND-Pol-CCA attack on the BF-PBE scheme can be converted into an IND-CCA attack on BasicPub^{hy}.

Lemma 4. *Let $\mathcal{A}^{\circ}$ be an* IND-PoI-CCA *adversary with advantage $Adv_{\mathcal{A}^{\circ}} \geq \varepsilon$ when attacking the* PB-PBE *scheme. Assume that $\mathcal{A}^{\circ}$ has running time $t_{\mathcal{A}^{\circ}}$ and makes at most q_c queries to oracle* CredGen-O*, q_d queries to oracle* PolDec-O *as well as q_0 queries to oracle H_0. Then, there exists an* IND-ID-CCA *adversary $\mathcal{A}^{\bullet}$ the advantage of which, when attacking the* BasicPub^{hy} *scheme, is such that $Adv_{\mathcal{A}^{\bullet}} \geq F(q_c, q_d, q_0, N, m_{\vee\wedge}, m_{\vee}, m_{\wedge}).\varepsilon$. Its running time is $t_{\mathcal{A}^{\bullet}} = O(t_{\mathcal{A}^{\circ}})$.*

Proof of Lemma 4 is similar to proof of Lemma 1 described in Appendix A. In the following, we first construct an IND-ID-CCA adversary $\mathcal{A}^{\bullet}$ that uses adversary $\mathcal{A}^{\circ}$ to mount an attack against the FullIdent scheme. The game between the challenger and algorithm $\mathcal{A}^{\bullet}$ starts with the Initialization stage which we describe below.

Initialization. On input of the security parameter k, the challenger first generates the public key $pk^{\star} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, R^{\star}, Q^{\star}, m^{\star}.n, H_1, H_2, H_3)$ such that $m^{\star} \in \{1, \ldots, m_{\vee\wedge}\}$ and $R^{\star} = s^{\star} \cdot P$, where $s^{\star} \in \mathbb{Z}_q^*$ is the private key corresponding to $pk^{\star}$. Upon receiving $pk^{\star}$, algorithm $\mathcal{A}^{\bullet}$ does the following:

1. Let $m^{\bullet} = m^{\star}$, then choose the values $m_i^{\bullet} \in \{1, \ldots, m_{\vee}\}$ and $m_{i,j}^{\bullet} \in \{1, \ldots, m_{\wedge}\}$ $\wr_{j=1}^{m_i^{\bullet}} \wr_{i=1}^{m^{\bullet}}$
2. Pick at random $\alpha_i^{\bullet} \in \{1, \ldots, m_{\vee\wedge}\}$ and $r_{\alpha_i^{\bullet}}, \omega_i^{\bullet} \in \mathbb{Z}_q^*$ $\wr_{i=1}^{m^{\bullet}}$
3. Pick at random $\beta_{i,j}^{\bullet} \in \{1, \ldots, m_{\vee}\}$ and $r_{\beta_{i,j}^{\bullet}}, \upsilon_{i,j}^{\bullet} \in \mathbb{Z}_q^*$ $\wr_{j=1}^{m_i^{\bullet}} \wr_{i=1}^{m^{\bullet}}$
4. Pick at random $l_{i,j,k}^{\bullet} \in \{1, \ldots, q_0\}$, $\gamma_{i,j,k}^{\bullet} \in \{1, \ldots, m_{\wedge}\}$ and $r_{\gamma_{i,j,k}^{\bullet}} \in \mathbb{Z}_q^*$ $\wr_{k=1}^{m_{i,j}^{\bullet}} \wr_{j=1}^{m_i^{\bullet}} \wr_{i=1}^{m^{\bullet}}$
5. Pick at random $\theta_{i,j,k}^{\bullet} \in \mathbb{Z}_q^* \wr_{k=2}^{m_{i,j}^{\bullet}} \wr_{j=1}^{m_i^{\bullet}}$ $\wr_{i=1}^{m^{\bullet}}$, then compute $\theta_{i,j,1}^{\bullet} = \sum_{k=2}^{m_{i,j}^{\bullet}} \theta_{i,j,k}^{\bullet}$ $\wr_{j=1}^{m_i^{\bullet}} \wr_{i=1}^{m^{\bullet}}$
6. Compute $\kappa_{i,j,k}^{\bullet} = ((\alpha_i^{\bullet} - 1).m_{\vee} + \beta_{i,j}^{\bullet}) - 1).m_{\wedge} + \gamma_{i,j,k}^{\bullet}$ and $r_{\kappa_{i,j,k}^{\bullet}} = r_{\gamma_{i,j,k}^{\bullet}} r_{\beta_{i,j}^{\bullet}} r_{\alpha_i^{\bullet}}$ $\wr_{k=1}^{m_{i,j}^{\bullet}} \wr_{j=1}^{m_i^{\bullet}} \wr_{i=1}^{m^{\bullet}}$
7. Choose a hash function: $\bar{H}_2^{\bullet} : \{1, \ldots, m_{\vee\wedge}\} \rightarrow \{0,1\}^n$
8. Define the function $\Delta^{\bullet} : \{0,1\}^{m^{\bullet}.n} \times \{1 \ldots, m^{\bullet}\} \rightarrow \{0,1\}^n$ which on input of a tuple (X, i) returns the i^{th} block of length n of the binary string X i.e. the bits from $(i-1).n+1$ to $i.n$ of X.

The interaction between algorithm $\mathcal{A}^{\bullet}$ and adversary $\mathcal{A}^{\circ}$ consists of five stages: Setup, Queries-1, Challenge, Queries-2 and Guess which we describe below.

Setup. Algorithm $\mathcal{A}^{\bullet}$ does the following: (1) Let $I^{\bullet} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, n_0, H_0^{\bullet}, H_1, H_2^{\bullet}, H_3)$ be the global information, where the oracles $H_0^{\bullet}$ and $H_2^{\bullet}$ are controlled by algorithm $\mathcal{A}^{\bullet}$ and the tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, n_0, H_1, H_3)$ is taken from $params^{\star}$, (2) Define the set of trusted authorities $\mathcal{T} = \{TA_1, \ldots, TA_N\}$ as follows: for $\kappa \in \{\kappa_{i,j,k}^{\bullet}\}$, the public key of TA_{κ} is $R_{\kappa} = r_{\kappa} \cdot R^{\star}$, whereas, for $\kappa \in \{1, \ldots, N\} \setminus \{\kappa_{i,j,k}^{\bullet}\}$, the public key of TA_{κ} is $R_{\kappa} = s_{\kappa} \cdot P$ for some randomly chosen $s_{\kappa} \in \mathbb{Z}_q^*$, (3) Give the global information $I^{\bullet}$ and the trusted authorities' public keys $R_{\kappa} \wr_{\kappa=1}^{N}$ to adversary $\mathcal{A}^{\circ}$.

Algorithm $\mathcal{A}^{\bullet}$ controls the random oracle $H_0^{\bullet}$ as follows: algorithm $\mathcal{A}^{\bullet}$ maintains a list of tuples $[A_{\iota}, H_{0,\iota}, \lambda_{\iota}]$ which we denote H_0^{list}. The list is initially empty. Assume that adversary $\mathcal{A}^{\circ}$ makes a query on assertion $A \in \{0,1\}^*$, then adversary $\mathcal{A}^{\bullet}$ responds as follows:

1. If A already appears on the list H_0^{list} in a tuple $[A_{\iota}, H_{0,\iota}, \lambda_{\iota}]$, then return $H_{0,\iota}$
2. If A does not appear on H_0^{list} and A is the $l_{i,j,1}^{\bullet}$-th distinct query to oracle $H_0^{\bullet}$, then compute $H_{0,l_{i,j,1}^{\bullet}} = r_{\gamma_{i,j,1}^{\bullet}}^{-1} \cdot ((r_{\beta_{i,j}^{\bullet}}^{-1} \upsilon_{i,j}^{\bullet} r_{\alpha_i^{\bullet}}^{-1} \omega_i^{\bullet}) \cdot Q^{\star} - \theta_{i,j,1}^{\bullet} \cdot P)$, return $H_{0,l_{i,j,1}^{\bullet}}$, and add $[A, H_{0,l_{i,j,1}^{\bullet}}, \text{null}]$ to H_0^{list}
3. If A does not appear on H_0^{list} and A is the $l_{i,j,k}^{\bullet}$-th distinct query to oracle $H_0^{\bullet}$ (for $k > 1$), then compute $H_{0,l_{i,j,k}^{\bullet}} = (r_{\gamma_{i,j,k}^{\bullet}}^{-1} \theta_{i,j,k}^{\bullet}) \cdot P$, return $H_{0,l_{i,j,k}^{\bullet}}$, and add the entry $[A, H_{0,l_{i,j,k}^{\bullet}}, r_{\gamma_{i,j,k}^{\bullet}}^{-1} \theta_{i,j,k}^{\bullet}]$ to H_0^{list}
4. Otherwise, pick at random $\lambda \in \mathbb{Z}_q^*$ such that $\lambda \cdot P$ does not appear on the list H_0^{list}, return $\lambda \cdot P$, and add the entry $[A, \lambda \cdot P, \lambda]$ to H_0^{list}

Algorithm $\mathcal{A}^{\bullet}$ controls the random oracle $H_2^{\bullet}$ as follows: on input of a tuple (G, i, j), algorithm $\mathcal{A}^{\bullet}$ returns the value $\Delta^{\bullet}(H_2(G^{\upsilon_{i,j}^{\bullet}{}^{-1} \omega_i^{\bullet}{}^{-1}}) \oplus \bar{H}_2^{\bullet}(j), i)$.

The policy $Pol_{\text{cr}} = \wedge_{i=1}^{m^\bullet} \vee_{j=1}^{m_i^\bullet} \wedge_{k=1}^{m_{i,j}^\bullet} \langle TA_{\kappa_{i,j,k}^\bullet}, A_{l_{i,j,k}^\bullet} \rangle$ is called the 'crucial' policy. Algorithm $\mathcal{A}^\bullet$ hopes that the 'target' policy Pol_{ch}, which will be chosen by adversary $\mathcal{A}^\circ$ in the Challenge stage of the IND-Pol-CCA game, will be equal to policy Pol_{cr}.

Queries-1. Adversary $\mathcal{A}^\circ$ performs a polynomial number of oracle queries adaptively.

Challenge. Once adversary $\mathcal{A}^\circ$ decides that the stage Queries-1 is over, it outputs two equal length messages M_0 and M_1 as well as a policy Pol_{ch} on which it wishes to be challenged. Algorithm $\mathcal{A}^\bullet$ responds as follows: (1) If $Pol_{\text{ch}} \neq Pol_{\text{cr}}$, then report failure and terminate (we refer to this event as $\mathcal{E}_{\text{ch}}$), (2) Otherwise, give the messages M_0, M_1 to the challenger who picks randomly $b \in \{0,1\}$ and returns a ciphertext $C^\star = (U, v^\star, W)$ representing the encryption of message M_b using the public key $pk^\star$. Upon receiving the challenger's response, compute the values $v_{i,j} = \Delta^\bullet(v^\star \oplus \bar{H}_2^\bullet(j), i)\ \wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$, then return the ciphertext $C_{\text{ch}} = (U, [[v_{i,j}]_{j=1}^{m_i^\bullet}]_{i=1}^{m^\bullet}, W)$.

For adversary $\mathcal{A}^\circ$, the ciphertext C_{ch} represents a correct encryption of message M_b according to policy Pol_{ch}. In fact, the ciphertext $C^\star$ is such that $U = H_1(M_b \| t) \cdot P$ (for some randomly chosen $t \in \{0,1\}^{m^\bullet . n}$), and $v^\star = t \oplus H_2(g^r)$ where $g = e(R^\star, Q^\star)$. Let $t_i = \Delta^\bullet(t, i)$, then the following holds

$$
\begin{aligned}
v_{i,j} &= \Delta^\bullet(t \oplus H_2(e(R^\star, Q^\star)^r) \oplus \bar{H}_2^\bullet(j), i) \\
&= \Delta^\bullet(t, i) \oplus \Delta^\bullet(H_2([e((r_{\beta_{i,j}^\bullet} r_{\alpha_i^\bullet}) \cdot R^\star, (r_{\beta_{i,j}^\bullet}^{-1} \upsilon_{i,j}^\bullet r_{\alpha_i^\bullet}^{-1} \omega_i^\bullet) \cdot Q^\star)^r]^{\upsilon_{i,j}^{\bullet\,-1} \omega_i^{\bullet\,-1}}) \oplus \bar{H}_2^\bullet(j), i) \\
&= t_i \oplus H_2^\bullet(e((r_{\beta_{i,j}^\bullet} r_{\alpha_i^\bullet}) \cdot R^\star, \sum_{k=1}^{m_{i,j}^\bullet} r_{\gamma_{i,j,k}^\bullet} H_{0, l_{i,j,k}^\bullet})^r, i, j) = t_i \oplus H_2^\bullet([\prod_{k=1}^{m_{i,j}^\bullet} e(R_{\kappa_{i,j,k}^\bullet}, H_{0, l_{i,j,k}^\bullet})]^r, i, j)
\end{aligned}
$$

Queries-2. Again, adversary $\mathcal{A}^\circ$ performs a polynomial number of oracle queries adaptively.

Guess. Algorithm $\mathcal{A}^\circ$ outputs a guess b' for b. Algorithm $\mathcal{A}^\bullet$ outputs b' as its guess for b.

The oracles CredGen-O and PolDec-O to which adversary $\mathcal{A}^\circ$ makes queries during Queries-1 and Queries-2 are described below. Without loss of generality, we assume that adversary $\mathcal{A}^\circ$ always makes the appropriate query on A to the random oracle $H_0^\bullet$ before making any query involving A to oracles CredGen-O and PolDec-O.

- **CredGen-O**. Assume that adversary $\mathcal{A}^\circ$ makes a query on a tuple (TA_κ, A). Let $[A_\iota, H_{0,\iota}, \lambda_\iota]$ be the tuple from H_0^{list} such that $A_\iota = A$, then algorithm $\mathcal{A}^\bullet$ responds as follows:
  1. If $\iota = l_{i,j,1}^\bullet$ and $\kappa \in \{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, then report failure and terminate (event $\mathcal{E}_{\text{cred}}$)
  2. If $\iota \neq l_{i,j,1}^\bullet$ and $\kappa \in \{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, then return $(r_\kappa \lambda_\iota) \cdot R^\star = (r_\kappa s^\star) \cdot H_{0,\iota}$
  3. If $\kappa \in \{1, \ldots, N\} \setminus \{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, then return $s_\kappa \cdot H_{0,\iota}$
- **PolDec-O**. Assume that adversary $\mathcal{A}^\circ$ makes an oracle query on a tuple $(C, Pol, \{j_1, \ldots, j_m\})$. Then, algorithm $\mathcal{A}^\bullet$ responds as follows:
  1. If $Pol \neq Pol_{\text{ch}}$ and Pol involves a condition $\langle TA_\kappa, A \rangle$ such that $\kappa \in \{\kappa_{i,j,k}^\bullet\}$ and $A \in \{A_{l_{i,j,1}^\bullet}\}$, then report failure and terminate (event $\mathcal{E}_{\text{dec}}$)
  2. If $Pol \neq Pol_{\text{ch}}$ and Pol does not involve any condition $\langle TA_\kappa, A \rangle$ such that $\kappa \in \{\kappa_{i,j,k}^\bullet\}$ and $A \in \{A_{l_{i,j,1}^\bullet}\}$, then do the following: (1) Run oracle CredGen-O multiple times until obtaining the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol)$, (2) Run algorithm PolDec on input the tuple $(C, Pol, \varsigma_{j_1,\ldots,j_m}(Pol))$ and return the resulting output back to adversary $\mathcal{A}^\circ$
  3. If $Pol = Pol_{\text{ch}}$, then do the following: let $C = (U, [[v_{i,j}]_{j=1}^{m_i^\bullet}]_{i=1}^{m^\bullet}, W)$, then compute the values $v_i^\bullet = v_{i,j_i} \oplus \bar{H}_2^\bullet(j_i)\ \wr_{i=1}^{m^\bullet}$, and make a decryption query to the challenger on ciphertext $C^\bullet = (U, v_1^\bullet \| \ldots \| v_{m^\bullet}^\bullet, W)$. Upon receiving the challenger's response, forward it to adversary $\mathcal{A}^\circ$

The analysis of the simulation described above is similar to the one in Appendix A.

□