Collusion-Free Policy-Based Encryption

Walid Bagga , Refik Molva

Institut Eurécom
Corporate Communications
2229, route des Crêtes B.P. 193
06904 Sophia Antipolis (France)
{bagga,molva}@eurecom.fr

Abstract. A policy-based encryption scheme allows a user to encrypt a message with respect to a credential-based policy formalized as monotone boolean expression written in standard normal form. The encryption is so that only a user having access to a qualified set of credentials for the policy is able to successfully decrypt the message. An inherent property of policy-based encryption is that in addition to the recipient an encrypted message is intended for, any collusion of credential issuers or end users who are able to collect a qualified set of credentials for the policy used to encrypt the message can decrypt it as well. In some applications, the collusion property may be acceptable or even useful. However, for most other applications it is undesirable. In this paper, we present a collusion-free policy-based encryption primitive, called policy-based public-key encryption. We provide precise definition for the new primitive as well as for the related security model. Then, we describe a concrete implementation using pairings over elliptic curves and prove its security in the random oracle model.

Keywords: Pairing-Based Cryptography, Provable Security, Authorization, Credentials

1 Introduction

Policy-based encryption, recently formalized in [4], allows to encrypt a message with respect to a credential-based policy formalized as monotone boolean expression written in standard normal form. The encryption is so that only a user that is compliant with the policy is able to decrypt the message. A policy involves conjunctions (logical AND operation) and disjunctions (logical OR operation) of conditions, where each condition is fulfilled by a digital credential representing the signature of a specific credential issuer on a set of statements about a certain entity. A user is thus compliant with a policy if and only if he has been issued a qualified set of credentials for the policy i.e. a set of credentials fulfilling the combination of conditions defined by the policy. More generally, policy-based encryption belongs to an emerging family of cryptographic schemes sharing the ability to integrate encryption with credential-based access structures. This ability allows for several interesting applications in different contexts including but not restricted to oblivious access control [4, 7, 14], trust negotiation [6, 9, 13], and cryptographic workflow [3].

Suppose that Alice wants to send a sensitive message to Bob, while requiring that Bob fulfills a specific credential-based policy in order for him to be authorized to read the message. In order to enforce her policy, Alice first encrypts her message according to her policy using a policy-based encryption algorithm, then she sends the resulting ciphertext to Bob. If Bob has access to a qualified set of credentials for Alice's policy, then he is compliant with the policy and can thus use his credentials to successfully decrypt the received message. An inherent property of the policy-based encryption primitive is that, in addition to Bob, any collusion of credential issuers or end users who are able to collect a qualified set of credentials for Alice's policy can decrypt the message as well. In some applications, where it is acceptable to assume that the credential issuers

are trusted for not colluding with each other to spy the end user's communications and that no end user is willing to share his credentials with other end users, the collusion property is not a problem. In other applications, collusion of end users is useful when collaboration is required to authorize access to sensitive information, while collusion of credential issuers may even be desirable for law enforcement. However, for other applications such as trust establishment in large-scale open environments like the Internet, the collusion property is undesirable.

In this paper, we present a collusion-free variance of the policy-based encryption primitive defined in [4], that we call *policy-based public key encryption*. The intuition behind our encryption primitive is as follows: we assume that each end user is associated to a public/private key pair. We suppose that no end user is willing to share his private key with the others, and thus keeps secret his valuable key in a secure storage system such as a smart card. Furthermore, we suppose that a credential delivered by a credential issuer is associated to the requester's public key, and that it is issued after checking that the requester possesses the corresponding private key. As opposed to the basic policy-based encryption primitive, our encryption algorithm takes as input, in addition to a credential-based policy, the public key of the recipient the encrypted message is intended for. The policy taken as input is fulfilled by qualified sets of credentials for which all the credentials are associated to the recipient's public key. Our decryption algorithm is such that, in order to successfully decrypt the message, one needs to have access to a qualified set of credentials for the policy as well as to the recipient's private key. Thus, our policy-based public-key encryption primitive prevents collusions of credential issuers by making the decryption algorithm involve a secret element (private key) held only by the recipient the encrypted message is intended for. Besides, our primitive prevents collusions of end users by associating all the credentials fulfilling the policy according to which a message is encrypted to the same public key, and making these credentials useful only in conjunction with the corresponding private key.

In the following, we present the related work found so far in the literature.

1.1 Related Work

As said before, policy-based encryption and policy-based public-key encryption belong to an emerging family of cryptographic schemes sharing the ability to integrate encryption with credential-based access control structures. This ability is mainly enabled by pairings over elliptic curves, and more particularly by the Boneh-Franklin identity-based encryption from bilinear pairings [5]. Note that identity-based encryption could be seen as a particular case of policy-based encryption. In fact, an identity-based encryption scheme corresponds to a policy-based encryption scheme for which policies are reduced to a single credential representing the signature of a centralized credential issuer (called private key generator) on the identity of an end user.

In [7], the authors present various applications of the use of multiple trusted authorities and multiple identities in the type of identity-based cryptography. They show how to perform encryption according to disjunctions and conjunctions of credentials. However, their solution remains restricted to a limited number of disjunctions. In [14], the author further pursues the ideas discussed in [7] and presents an elegant and efficient mechanism to perform access control based on encryption with respect to monotone boolean expressions written in standard normal forms. The proposed solution remains limited to credentials generated by a centralized trusted authority. Furthermore, it lacks adequate security arguments. In [4], the authors provide a further generalization of [14] by considering credentials that might be generated by independent credential issuers. They formalize the concept of policy-based cryptography and provide precise definitions for policy-based encryption and policy-based signature primitives. Furthermore, they show how such primitives could 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. Unfortunately, the presented schemes lack formal security analysis as for [14].

In [9], the authors introduce hidden credentials as a solution to perform privacy-enabled trust negotiation. Their solution uses the Boneh-Franklin identity-based encryption scheme [5] and relies on onion-like encryption and multiple encryption operations to deal, respectively, with conjunctions and disjunctions of credentials. Such approach remains inefficient in terms of both computational costs and bandwidth consumption (ciphertext size), especially when authorization structures become complex. In [6], 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 under the identity-based security models defined in [5].

Although used in application scenarios with different security requirements, the encryption schemes presented above share the fact that they allow to encrypt a message according to a credential-based policy so that only the users having access to a qualified set of credentials for the policy are able to successfully decrypt the message. While the schemes of [6, 9] consider policies formalized as monotone boolean expressions written as general conjunctions and disjunctions of atomic terms, the schemes of [4, 14] consider the ones written in standard normal forms. All the presented schemes are based on the Boneh-Franklin identity-based encryption primitive described in [5], from which they inherit the collusion property. In fact, the Boneh-Franklin scheme suffers from the key-escrow property i.e. the credential issuer is able to decrypt the confidential messages intended for the end users. As for the collusion property faced by policy-based encryption, the key-escrow property might be necessary in some contexts, especially within organizations, for monitoring and law enforcement. However, in most applications, it is undesirable.

In [1], the authors describe a modification of the Boneh-Franklin encryption scheme that allows to avoid the key-escrow problem. Their primitive, called certificateless public-key encryption, requires each end user to have a public key. The encryption of a message is performed with respect to the identity of the recipient as well as with respect to his public key. The decryption algorithm requires both the recipient's private key and his identity credential. In [3], the authors consider general access structures and use a similar technique to achieve the policy-based encryption functionality while avoiding the collusion property. Their scheme could be seen as the collusion-free variance of the encryption scheme proposed in [6]. They underline the fact that their scheme supports cryptographic workflow, which is a feature inherited from the Boneh-Franklin encryption primitive and supported by the policy-based encryption primitive as well. They define formal security models to support their encryption primitive. Their 'recipient security model' considers indistinguishability against chosen plaintext attacks, where the adversary does not have access to the decryption oracle. Security against the stronger chosen ciphertext attacks is left as an open research problem.

1.2 Contributions and Outline of the Paper

In this paper, we define a new policy-based cryptographic primitive, called policy-based public-key encryption, and describe a provably secure concrete implementation based on bilinear pairings over elliptic curves. Our primitive allows to overcome the collusion problem faced by the original policy-based encryption primitive defined in [4]. We use a technique similar to the one used in [1] to overcome the key-escrow problem from which may suffer the identity-based encryption primitive defined in [5]. The escrow-free encryption scheme proposed in [3] may be considered as a policy-based public-key encryption scheme when applied to policies written in standard normal forms. Some may consider that restricting our scheme to standard normal forms is a limitation compared to the scheme of [3] which deals with general-form policies. We argue that this is not so, as in real-world scenarios security policies are typically written in standard normal forms. For

example, the Web Service policy languages WS-Policy and WSPL consider policies converted to the standard disjunctive normal form. Our concrete scheme improves the performance of the key-escrow encryption scheme of [3] (when applied to standard-form policies) both in terms of computational cost and bandwidth consumption (size of the resulting ciphertext). Furthermore, we prove the security of our scheme against chosen ciphertext attacks as opposed to the approach of [3] that consider the weaker chosen plaintext attacks.

The rest of the paper is organized as follows: in Section 2, we first set the context for our encryption primitive including the terminology, the notation and the policy model. Then, we provide a precise definition for policy-based public-key encryption schemes as well as for the related security model. The latter adapts the strong security notion of indistinguishability against chosen ciphertext attacks to the specific features of the new policy-based cryptographic primitive. In Section 3, we describe a concrete policy-based public-key encryption primitive based on bilinear pairings over elliptic curves. Our scheme is a modification of the original policy-based encryption scheme described in [4] that integrates the public and private keys in the encryption and decryption algorithms respectively. As opposed to the scheme presented in [4] which lacks formal security analysis, we provide reductionist security arguments for our scheme in the random oracle model.

2 Definitions

2.1 Setting the Context

We consider a public key infrastructure where each end user holds a pair of keys (pk_u, sk_u). An end user is identified by his public key pk_u. The public key does not has to be bound to the end user's name/identity (through public-key certification) as for standard PKI systems such as X.509. In fact, in large-scale open environments, the identity of an end user is rarely of interest to determining whether the end user could be trusted or authorized to conduct some sensitive transactions. Instead, statements about the end user such as attributes, properties, capabilities and/or privileges are more relevant. The validity of such statements is checked and certified by trusted entities called credential issuers through a digital signature procedure.

We consider a set of credential issuers $I = \{I_1, \ldots, I_N\}$, where the public key of I_κ, for $\kappa \in \{1, \ldots, N\}$, is denoted R_κ while the corresponding master key is denoted s_κ. We assume that a trustworthy value of the public key of each of the credential issuers is known by the end users. Any credential issuer $I_\kappa \in I$ may be asked by an end user to issue a credential corresponding to a set of statements. The requested credential is basically the digital signature of the credential issuer on an assertion denoted A^{pk_u}. The assertion contains, in addition to the set of statements, the end user's public key pk_u as well as a set of additional information such as the validity period of the credential. As the representation of assertions is out of the scope of this paper, they will simply be encoded as binary strings.

Upon receiving a request for generating a credential on assertion A^{pk_u}, a credential issuer I_κ first checks the fact that the requester has access to the private key sk_u associated to pk_u. Then, the credential issuer checks the validity of the assertion A^{pk_u}. If it is valid, then I_κ executes a credential generation algorithm and returns a credential denoted $\varsigma(R_\kappa, A^{pk_u})$. Otherwise, I_κ returns an error message. Upon receiving the credential $\varsigma(R_\kappa, A^{pk_u})$, the end user may check its integrity using I_κ's public key R_κ. The process of checking the validity of a set of statements about a certain entity is out of the scope of this paper.

We consider credential-based policies formalized as monotone boolean expressions involving conjunctions (AND / $\wedge$) and disjunctions (OR / $\vee$) of credential-based conditions. A credential-based condition is defined through a pair $\langle I_\kappa, A^{pk_u} \rangle$ specifying an assertion $A^{pk_u} \in \{0,1\}^*$ (about an end user whose public key is pk_u)

and a credential issuer $I_\kappa \in I$ that is trusted to check and certify the validity of A^{pk_u}. An end user whose public key is pk_u fulfills the condition $\langle I_\kappa, A^{pk_u} \rangle$ if and only if he has been issued the credential $\varsigma(R_\kappa, A^{pk_u})$.

We consider policies written in standard normal forms, i.e. written either in conjunctive normal form (CNF) or in disjunctive normal form (DNF). In order to address the two standard normal forms, we use the conjunctive-disjunctive normal form (CDNF) introduced in [14]. Thus, a policy denoted Pol^{pk_u} is written as follows:

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

Under the CDNF notation, policies written in CNF correspond to the case where $\{m_{i,j} = 1\}_{i,j}$, while policies written in DNF correspond to the case where $m = 1$.

Let $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$ denote the set of credentials $\{\{\varsigma(R_{\kappa_{i,j_i,k}}, A_{i,j_i,k}^{pk_u})\}_{k=1}^{m_{i,j_i}}\}_{i=1}^{m}$, for some $\{j_i \in \{1,\ldots,m_i\}\}_{i=1}^{m}$. Then, $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$ is a qualified set of credentials for policy Pol^{pk_u}.

2.2 Policy-Based Public-Key Encryption

A policy-based public-key encryption scheme (denoted in short PB-PKE) is specified by six algorithms: *System-Setup*, *Issuer-Setup*, *User-Setup*, *CredGen*, *Encrypt* and *Decrypt*, which we describe below.

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

Issuer-Setup. This algorithm generates a random master key s_κ and the corresponding public key R_κ for credential issuer $I_\kappa \in I$.

User-Setup. This algorithm generates a random private key sk_u and the corresponding public key pk_u.

CredGen. On input of the public key R_κ of a credential issuer $I_\kappa \in I$ and an assertion $A^{pk_u} \in \{0,1\}^*$, this algorithm returns the credential $\varsigma(R_\kappa, A^{pk_u})$.

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

Decrypt. On input of a ciphertext $C \in \mathcal{C}$, a pair of keys (pk_u, sk_u), a policy Pol^{pk_u} and a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$, 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:

$$C = Encrypt(M, Pol^{pk_u}, pk_u) \;\Rightarrow\; Decrypt(C, Pol^{pk_u}, pk_u, sk_u, \varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})) = M$$

Finally, we define $\varphi_{j_1,\ldots,j_m}(C, pk_u, Pol^{pk_u})$ to be the information from C that is required to correctly perform the decryption of C with respect to policy Pol^{pk_u} and public key pk_u using the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$. A concrete example is given when describing our PB-PKE scheme. Such information is used in the specification of the security model associated to the policy-based public-key encryption primitive.

2.3 Security Model

Our security model for PB-PKE schemes follows the following reasoning: the standard acceptable notion of security for public key encryption schemes is indistinguishability against chosen ciphertext attacks (IND-CCA). Hence, it is natural to require that a PB-PKE scheme also satisfies this strong notion of security. However,

the definition of this security notion must be adapted to the policy-based setting. A PB-PKE scheme is such that a user should not be able to decrypt a message if he does not fulfill the policy according to which the message was encrypted or if he does not have access to the private key corresponding to the public key used to encrypt the message. Assume, for instance, that a user Alice wants to send a sensitive message to a user Bob whose public key is pk_b. Moreover, assume that Alice wants to be sure that Bob is compliant with a specific policy Pol^{pk_b} in order for Bob to be able to read the message. Thus, Alice uses a PB-PKE scheme to encrypt her message using Bob's public key pk_b according to her policy Pol^{pk_b}. Two attack scenarios should be considered:

- In the first scenario, a third user Charlie that has somehow access to a qualified set of credentials for policy Pol^{pk_b} tries to decrypt the intercepted message. For example, Charlie may represent a collusion of the different credential issuers specified by Pol^{pk_b}. As Charlie has not access to Bob's private key sk_b, he must not be able to successfully achieve the decryption. Because Charlie is not the legitimate recipient of the message he will be called *Outsider*.
- In the second scenario, the user Bob (who has access to the private key sk_b) does not have access to a qualified set of credentials for policy Pol^{pk_b} and tries to illegally decrypt the message. As Bob does not fulfill Alice's policy, he must not be able to successfully decrypt the message. As opposed to the Outsider adversary, Bob will be called *Insider*.

Our security model is defined in terms of an interactive game played between a challenger and an adversary, where the adversary can be either Insider or Outsider. The game consists of five stages: *Setup*, *Phase-1*, *Challenge*, *Phase-2* and *Guess*, which we describe below.

- ***Setup***. On input of a security parameter k, the challenger does the following: (1) Run algorithm *System-Setup* to obtain the system public parameters $\mathcal{P}$ which are given to the adversary, (2) Run algorithm *Issuer-Setup* once or multiple times to obtain a set of credential issuers $I = \{I_1, \ldots, I_N\}$, (3) Run algorithm *User-Setup* to obtain a public/private key pair $(pk_{\text{ch}}, sk_{\text{ch}})$. Depending on the type of the adversary, the challenger does the following: If the adversary is an Outsider, then the challenger gives to the adversary the public keys as well as the master keys of the credential issuers included in I. Furthermore, the challenger gives the public key pk_{ch} to the adversary while keeping secret the corresponding private key sk_{ch}. However, if the adversary is an Insider, then the challenger just gives to the adversary, in addition to the pair of keys $(pk_{\text{ch}}, sk_{\text{ch}})$, the public keys of the credential issuers included in I while keeping secret the corresponding master keys.
- ***Phase-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***. This stage occurs when the adversary decides that the *Phase-1* stage is over. The adversary, be it Insider or Outsider, gives to the challenger two equal length messages M_0, M_1 and a policy $Pol_{\text{ch}}^{pk_{\text{ch}}}$ on which he wishes to be challenged. The challenger picks at random $b \in \{0,1\}$, then runs algorithm *Encrypt* on input of the tuple $(M_b, pk_{\text{ch}}, Pol_{\text{ch}}^{pk_{\text{ch}}})$, and returns the resulting ciphertext C_{ch} to the adversary.
- ***Phase-2***. The adversary performs again a polynomial number of adaptive oracle queries.
- ***Guess***. The adversary outputs a guess b', and wins the game if $b = b'$.

During the *Phase-1* and *Phase-2* stages, the adversary may perform queries to two oracles controlled by the challenger. On one hand, a credential generation oracle denoted ***CredGen-O***. On the other hand, a decryption oracle denoted ***Decrypt-O***. While the oracles are executed by the challenger, their input is specified by the adversary. The two oracles are defined as follows:

– ***CredGen-O***. On input of a credential issuer $I_\kappa \in I$ and an assertion $A^{pk_u} \in \{0,1\}^*$, run algorithm *CredGen* on input of the tuple (I_κ, A^{pk_u}) and return the resulting credential $\varsigma(R_\kappa, A^{pk_u})$. Note that an Outsider does not need to perform queries to this oracle as he has access to the credential issuers' master keys. Besides, an Insider is not allowed to obtain a qualified set of credentials for the policy $Pol_{\text{ch}}^{pk_{\text{ch}}}$ which he is challenged on.
– ***Decrypt-O***. On input of a ciphertext $C \in \mathcal{C}$, a policy Pol^{pk_u} and a set of indices $\{j_1, \ldots, j_m\}$, first run algorithm *CredGen* multiple times to obtain the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_{\text{ch}}})$, then run algorithm *Decrypt* on input of the tuple $(C, pk_{\text{ch}}, sk_{\text{ch}}, Pol^{pk_{\text{ch}}}, \varsigma_{j_1,\ldots,j_m}(Pol^{pk_{\text{ch}}}))$, and return the resulting output. Note that an adversary, be it Insider or Outsider, cannot perform a query to oracle *Decrypt-O* on a tuple $(C, Pol_{\text{ch}}^{pk_{\text{ch}}}, \{j_1, \ldots, j_m\})$ such that $\varphi_{j_1,\ldots,j_m}(C, pk_{\text{ch}}, Pol_{\text{ch}}^{pk_{\text{ch}}}) = \varphi_{j_1,\ldots,j_m}(C_{\text{ch}}, pk_{\text{ch}}, Pol_{\text{ch}}^{pk_{\text{ch}}})$.

The game described above is denoted *IND-Pol-CCA*$^{\mathsf{X}}_{\mathsf{PK}}$, where $\mathsf{X} = \mathsf{I}$ for Insider adversaries and $\mathsf{X} = \mathsf{O}$ for Outsider adversaries. A formal definition of chosen ciphertext security for PB-PKE 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 $\mathcal{A}^{\mathsf{X}}$ in the IND-Pol-CCA$^{\mathsf{X}}_{PK}$ game is defined to be the quantity $Adv_{\mathcal{A}^{\mathsf{X}}} = |Pr[b = b'] - \frac{1}{2}|$. A* PB-PKE *scheme is IND-Pol-CCA$^{\mathsf{X}}_{PK}$ secure if no probabilistic polynomial time adversary has a non-negligible advantage in the IND-Pol-CCA$^{\mathsf{X}}_{PK}$ game.*

Note. Our security model could be viewed as an extension to the policy-based public-key setting of the IND-ID-CCA model defined in [5]. 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 public-key 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 $(C_{\text{ch}}, Pol_{\text{ch}}^{pk_{\text{ch}}})$ is not sufficient anymore. In fact, we may have tuples such that $(C, Pol^{pk_{\text{ch}}}) \neq (C_{\text{ch}}, Pol_{\text{ch}}^{pk_{\text{ch}}})$ while $\varphi_{j_1,\ldots,j_m}(C, Pol^{pk_{\text{ch}}}) = \varphi_{j_1,\ldots,j_m}(C_{\text{ch}}, Pol_{\text{ch}}^{pk_{\text{ch}}})$. Decryption queries on such tuples should then be forbidden as well. $\diamond$

3 Our PB-PKE Scheme

3.1 Description

Before describing our PB-PKE scheme, we define algorithm *BDH-Setup* as follows:

BDH-Setup. Given 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 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-1. 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-2. 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 [10, 15] for further details. $\diamond$

Our PB-PKE scheme consists of the algorithms described below.

System-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 $\mathcal{M} = \{0,1\}^n$, $\mathcal{K} = \mathbb{G}_1$ 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 $H_3 : \{0,1\}^* \rightarrow \{0,1\}^n$
4. Let $\mathcal{P} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_0, H_1, H_2, H_3)$.

Issuer-Setup. Let $I = \{I_1, \ldots, I_N\}$ be a set of credential issuers. Each credential issuer $I_\kappa \in I$ picks at random a secret master key $s_\kappa \in \mathbb{Z}_q^*$ and publishes the corresponding public key $R_\kappa = s_\kappa \cdot P$.

User-Setup. This algorithm picks at random a private key $sk_u \in \mathbb{Z}_q^*$ and computes the corresponding public key $pk_u = sk_u \cdot P$.

CredGen. On input of issuer $I_\kappa \in I$ and assertion $A^{pk_u} \in \{0,1\}^*$, this algorithm outputs $\varsigma(R_\kappa, A^{pk_u}) = s_\kappa \cdot H_0(A^{pk_u})$.

Encrypt. On input of message $M \in \mathcal{M}$, public key pk_u and policy Pol^{pk_u}, do the following:

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

The intuition behind the encryption algorithm is as follows: each conjunction of conditions $\wedge_{k=1}^{m_{i,j}} \langle I_{\kappa_{i,j,k}}, A_{i,j,k}^{pk_u} \rangle$ is first associated to a mask $\mu_{i,j}$ that depends not only on the different credentials related to the specified conditions but also on the specified public key. Then, for each index $i \in \{1, \ldots, m\}$, a randomly chosen intermediate key t_i is associated to the disjunction $\vee_{j=1}^{m_i} \wedge_{k=1}^{m_{i,j}} \langle I_{\kappa_{i,j,k}}, A_{i,j,k}^{pk_u} \rangle$. Finally, each intermediate key 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 t_i. In order to be able to retrieve the encrypted message, an entity needs to retrieve all the intermediate keys t_i using not only a qualified set of credentials for policy Pol^{pk_u}, but also the private key sk_u corresponding to pk_u.

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

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

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

3.2 Consistency and Efficiency

The algorithms described above satisfy the standard consistency constraint. In fact, we have, on one hand, $\tilde{K} = sk_u \cdot U = sk_u \cdot (r \cdot P) = r \cdot (sk_u \cdot P) = r \cdot pk_u$. On the other hand, the following holds

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

The essential operation in pairing-based cryptography is pairing computations. Although such operation can be optimized, it still have to be minimized. In Table 1, we provide the computational costs of our encryption and decryption algorithms in terms of pairing computations as well as the size of the resulting ciphertext. Note that l_1 denotes the bit-length of the bilinear representation of an element of group $\mathbb{G}_1$.

	Encryption	Decryption	Ciphertext Size
Our PB-PKE scheme	$\sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}$	m	$l_1 + (\sum_{i=1}^{m} m_i).n + n$
The scheme of [3]	$\sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}$	$\sum_{i=1}^{m} m_{i,j_i}$	$l_1 + (\sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}).n + n$

Table 1. Performance of our PB-PKE scheme compared with the scheme of [3]

In Table 1, we provide the performance of the key-escrow scheme of [3] when applied to policies written in standard normal forms following the notation defined in Section 2. While the encryption algorithms require the same amount of pairing computations, our decryption algorithm more efficient as $m_{i,j_i} \geq 1$ for $i = 1, \ldots, m$. Furthermore, as $m_{i,j} \geq 1$ for $j = 1, \ldots, m_i$ and $i = 1, \ldots, m$, the size of the ciphertexts resulting from our scheme is at least as short as the one of the ciphertexts produced by the scheme of [3].

Note. As for standard asymmetric encryption schemes, PB-PKE schemes are much less efficient than symmetric encryption schemes. In practice, they should be used to exchange the symmetric (session) keys that are used for bulk encryption.

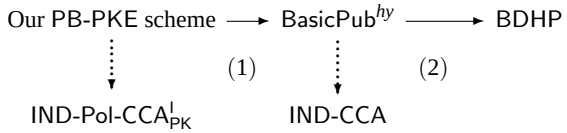
3.3 Security

In the following, we show respectively that our PB-PKE scheme is both IND-Pol-CCA$^I_{PK}$ and IND-Pol-CCA$^O_{PK}$ secure in the random oracle model.

Notation. Given the notation used in Section 2, the maximum values that the quantities m, m_i and $m_{i,j}$ 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 system setup. $\diamond$

Theorem 1. *Our* PB-PKE *scheme is* IND-Pol-CCA$^I_{PK}$ *secure in the random oracle model under the assumption that BDHP is hard.*

Proof. Theorem 1 follows from a sequence of reduction arguments that are summarized in the following diagram:



1. Lemma 1 shows that an IND-Pol-CCA$^{I}_{PK}$ attack on our PB-PKE scheme can be converted into an IND-CCA attack on the BasicPub^{hy} algorithm defined in [5].
2. In [5], algorithm BasicPub^{hy} is shown to be IND-CCA secure in the random oracle model under the assumption that BDHP is hard.

Lemma 1. *Let $\mathcal{A}^{\circ}$ be an* IND-Pol-CCA$^{I}_{PK}$ *adversary with advantage $Adv_{\mathcal{A}^{\circ}} \geq \varepsilon$ when attacking our* PB-PKE *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* Decrypt-O *as well as q_0 queries to oracle H_0. Then, there exists an* IND-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 1 is given in Appendix A.

Note-1. Lemma 1 stated below uses the quantity $F(q_c, q_d, q_0, N, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ defined as follows:

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

where $\Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = \Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) - \Upsilon(Nq_0 - (m_{\vee\wedge} m_{\vee})^2, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) - 1$.

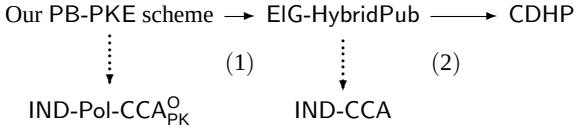
Computing $F(.)$ relies on computing the quantity $\Upsilon(X, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$, which is defined to be the total number of 'minimal' (reduced) policies written in CDNF, given the upper-bounds $(m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ and X possible credential-based conditions. Computing $\Upsilon(X, m_{\vee\wedge}, m_{\vee}, m_{\wedge})$ is similar, but not exactly the same as the problems of computing the number of monotone boolean functions of *n* variables (Dedekind's Problem [12]) and computing the number of antichains on a set $\{1, \ldots, n\}$ [11]. 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. As we do not address the practical security of our scheme in this paper, we do not elaborate more on the details. $\diamond$

Note-2. In the particular case where $N = m_{\vee\wedge} = m_{\vee} = m_{\wedge} = 1$, we have $\Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = 0$ and $\Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = q_0$. In this case, our PB-PKE scheme when attacked by the Insider adversary is equivalent to the FullIdent scheme of [5]. Note that our results match Result 5 of [8]. In fact, our reductionist security proof follows a strategy similar to the one used in [8]. $\diamond$

Note-3. The result of our security reduction remains theoretical. The function $F(.)$ depends exponentially on the policy size bounds which is not acceptable in practice. We are currently working on improving the tightness of our reduction in order to determine exact security arguments for real-world scenarios. $\diamond$

Theorem 2. *Our* PB-PKE *scheme is* IND-Pol-CCA$^{O}_{PK}$ *secure in the random oracle model under the assumption that CDHP is hard.*

Proof. Theorem 2 follows from two reduction arguments that are summarized in the following diagram:



1. Lemma 2 shows that an IND-Pol-CCA$^{\mathsf{O}}_{\mathsf{PK}}$ attack on our PB-PKE scheme can be converted into an IND-CCA attack on the EIG-HybridPub algorithm defined in [2].
2. In [2], algorithm EIG-HybridPub^{hy} is shown to be IND-CCA secure in the random oracle model under the assumption that CDHP is hard.

Lemma 2. *Let $\mathcal{A}^{\circ}$ be an* IND-Pol-CCA$^{\mathsf{O}}_{PK}$ *adversary with advantage $Adv_{\mathcal{A}^{\circ}} \geq \varepsilon$ when attacking our* PB-PKE *scheme. Then, there exists an* IND-CCA *adversary $\mathcal{A}^{\bullet}$ the advantage of which, when attacking the* EIG-HybridPub *scheme, is such that $Adv_{\mathcal{A}^{\bullet}} \geq \varepsilon$. Its running time is $t_{\mathcal{A}^{\bullet}} = O(t_{\mathcal{A}^{\circ}})$.*

Proof of Lemma 2 is given in Appendix B.

4 Conclusion

In this paper, we presented a collusion-free policy-based encryption primitive. We provided formal definitions for the new primitive and described a concrete implementation using bilinear pairings over elliptic curves. We defined a strong security model for our primitive following the notion of indistinguishability against chosen ciphertext attacks, and proved the security of our pairing-based scheme in the random oracle model. The goal of the new primitive is to overcome the weakness of the original policy-based encryption primitive defined in [4] when used in application scenarios for which collusions of credential issuers and end users are undesirable. The key-escrow encryption scheme presented in [3] allows to achieve the same security goals when applied to policies written in standard normal forms. Our proposal improves the scheme of [3] in terms of both performance and formal security analysis. Our security analysis remains theoretical and the results of our reductionist proof are unacceptable for practical use of our primitive. We are currently working on improving the tightness of our reduction and determining exact security parameters for real-world scenarios. A target application for our primitive is trust establishment and negotiation in large-scale open environments.

References

1. S. Al-Riyami and K. Paterson. Certificateless public key cryptography. In *ASIACRYPT*, pages 452–473. Springer-Verlag, 2003.
2. S.S. Al-Riyami. Cryptographic schemes based on elliptic curve pairings. Ph.D. Thesis, Royal Holloway, University of London, 2004.
3. 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/.
4. 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.
5. 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.
6. R. Bradshaw, J. Holt, and K. Seamons. Concealing complex policies with hidden credentials. Cryptology ePrint Archive, Report 2004/109, 2004. http://eprint.iacr.org/.
7. 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.
8. D. Galindo. Boneh-franklin identity based encryption revisited. To appear in Proceedings of 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005).
9. 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.

10. 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.
11. J. Kahn. Entropy, independent sets and antichains: a new approach to dedekind's problem. In *Proc. Amer. Math. Soc. 130*, pages 371–378, 2002.
12. D. Kleitman. On dedekind's problem: the number of monotone boolean functions. In *Proc. Amer. Math. Soc. 21*, pages 677–682, 1969.
13. 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.
14. N. Smart. Access control using pairing based cryptography. In *Proceedings CT-RSA 2003*, pages 111–121. Springer-Verlag LNCS 2612, April 2003.
15. Y. Yacobi. A note on the bilinear diffie-hellman assumption. Cryptology ePrint Archive, Report 2002/113, 2002. http://eprint.iacr.org/.

A Proof of Lemma 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'. $\diamond$

We construct an IND-CCA adversary $\mathcal{A}^\bullet$ that uses adversary $\mathcal{A}^\circ$ to mount an attack against the BasicPub^{hy} algorithm of [5]. 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 BasicPub^{hy} public key $PK^\star = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, R^\star, Q^\star, m^\star.n, H_0, H_1, H_2, H_3)$ 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 two hash functions: $\bar{H}_2^\bullet : \{1, \ldots, m_{\vee\wedge}\} \to \{0,1\}^n$ and $\hat{H}_2^\bullet : \mathbb{G}_1 \to \{0,1\}^{m^\bullet.n}$
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.

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*, *Phase-1*, *Challenge*, *Phase-2* and *Guess* which we describe below.

Setup. Algorithm $\mathcal{A}^\bullet$ does the following: (1) Let $\mathcal{P}^\bullet = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_0^\bullet, H_1, H_2^\bullet, H_3)$ the system public parameters, where the oracles $H_0^\bullet$ and $H_2^\bullet$ are controlled by algorithm $\mathcal{A}^\bullet$ and the tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_1, H_3)$ is taken from $PK^\star$, (2) Define the the set of credential issuers $I = \{I_1, \ldots, I_N\}$ as follows: for $\kappa = \kappa_{i,j,k}^\bullet$, the public key of $I_{\kappa_{i,j,k}^\bullet}$ is $R_{\kappa_{i,j,k}^\bullet} = r_{\kappa_{i,j,k}^\bullet} \cdot R^\star$, whereas, for $\kappa \in \{1, \ldots, N\} \setminus \{\kappa_{i,j,k}^\bullet\}_{i,j,k}$, the public key of I_κ is

$R_\kappa = s_\kappa \cdot P$ for some randomly chosen $s_\kappa \in \mathbb{Z}_q^*$, (3) Generate at random a pair of keys $(pk_{\mathrm{ch}}, sk_{\mathrm{ch}})$ and give it to the adversary, (4) Give the public parameters $\mathcal{P}^\bullet$ and the trusted authorities' public keys $R_\kappa \,\wr_{\kappa=1}^{N}$ to adversary $\mathcal{A}^\circ$.

Note. For $\kappa \in \{\kappa^\bullet_{i,j,k}\}_{i,j,k}$, the master key of I_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 $A^{pk_u} \in \{0,1\}^*$, then adversary $\mathcal{A}^\bullet$ responds as follows:

1. If A^{pk_u} 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^{pk_u} does not appear on H_0^{list} and A^{pk_u} is the $l^\bullet_{i,j,1}$-th distinct query to oracle $H_0^\bullet$, then compute $H_{0,l^\bullet_{i,j,1}} = r^{-1}_{\gamma^\bullet_{i,j,1}} \cdot ((r^{-1}_{\beta^\bullet_{i,j}} \upsilon^\bullet_{i,j} r^{-1}_{\alpha^\bullet_i} \omega^\bullet_i) \cdot Q^\star - \theta^\bullet_{i,j,1} \cdot P)$, return $H_{0,l^\bullet_{i,j,1}}$, and add $[A^{pk_u}, H_{0,l^\bullet_{i,j,1}}, \mathsf{null}]$ to H_0^{list}
3. If A^{pk_u} does not appear on H_0^{list} and A^{pk_u} is the $l^\bullet_{i,j,k}$-th distinct query to oracle $H_0^\bullet$ (for $k > 1$), then compute $H_{0,l^\bullet_{i,j,k}} = (r^{-1}_{\gamma^\bullet_{i,j,k}} \theta^\bullet_{i,j,k}) \cdot P$, return $H_{0,l^\bullet_{i,j,k}}$, and add the entry $[A^{pk_u}, H_{0,l^\bullet_{i,j,k}}, r^{-1}_{\gamma^\bullet_{i,j,k}} \theta^\bullet_{i,j,k}]$ 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 $[A^{pk_u}, \lambda \cdot P, \lambda]$ to H_0^{list}

Note. The simulated oracle $H_0^\bullet$ is such that $(r^{-1}_{\beta^\bullet_{i,j}} \upsilon^\bullet_{i,j} r^{-1}_{\alpha^\bullet_i} \omega^\bullet_i) \cdot Q^\star = \sum_{k=1}^{m^\bullet_{i,j}} r_{\gamma^\bullet_{i,j,k}} H_{0,l^\bullet_{i,j,k}} \,\wr_{i,j}$

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

The policy $Pol_{\mathrm{cr}} = \wedge_{i=1}^{m^\bullet} \vee_{j=1}^{m^\bullet_i} \wedge_{k=1}^{m^\bullet_{i,j}} \langle I_{\kappa^\bullet_{i,j,k}}, A_{l^\bullet_{i,j,k}} \rangle$ is called the 'crucial' policy. Algorithm $\mathcal{A}^\bullet$ hopes that the 'target' policy $Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$, which will be chosen by adversary $\mathcal{A}^\circ$ in the *Challenge* stage of the $\mathsf{IND\text{-}Pol\text{-}CCA}^I_{PK}$ game, will be equal to policy Pol_{cr}.

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

Challenge. Once adversary $\mathcal{A}^\circ$ decides that Phase-1 is over. Then, he outputs two equal length messages M_0 and M_1 as well as a policy $Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$ on which he wishes to be challenged. Algorithm $\mathcal{A}^\bullet$ responds as follows: (1) If $Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}} \neq Pol_{\mathrm{cr}}$, then report failure and terminate (we refer to this event as $\mathcal{E}_{\mathsf{ch}}$), (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 $\mathsf{BasicPub}^{hy}$ encryption of message M_b using the public key $PK^\star$. Upon receiving the challenger's response, compute the values $v_{i,j} = \Delta^\bullet(\hat{H}_2^\bullet(pk_{\mathrm{ch}}) \oplus v^\star \oplus \bar{H}_2^\bullet(j), i) \,\wr_{j=1}^{m^\bullet_i} \wr_{i=1}^{m^\bullet}$, then return the ciphertext $C_{\mathrm{ch}} = (U, [[v_{i,j}]_{j=1}^{m^\bullet_i}]_{i=1}^{m^\bullet}, W)$ to adversary $\mathcal{A}^\circ$.

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 $U = H_1(M_b \| t) \cdot P$, $W = M_b \oplus H_3(t)$ (for some randomly chosen $t \in \{0,1\}^{m^\bullet \cdot n}$), 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

$$
\begin{aligned}
v_{i,j} &= \Delta^\bullet(\hat{H}_2^\bullet(pk_{\mathrm{ch}}) \oplus t \oplus H_2(e(R^\star, Q^\star)^r) \oplus \bar{H}_2^\bullet(j), i) \\
&= \Delta^\bullet(t, i) \oplus \Delta^\bullet(\hat{H}_2^\bullet(pk_{\mathrm{ch}}) \oplus H_2([e((r_{\beta^\bullet_{i,j}} r_{\alpha^\bullet_i}) \cdot R^\star, (r^{-1}_{\beta^\bullet_{i,j}} \upsilon^\bullet_{i,j} r^{-1}_{\alpha^\bullet_i} \omega^\bullet_i) \cdot Q^\star)^r]^{\upsilon^\bullet_{i,j}{}^{-1} \omega^{\bullet\,-1}_i}) \oplus \bar{H}_2^\bullet(j), i) \\
&= t_i \oplus H_2^\bullet(e((r_{\beta^\bullet_{i,j}} r_{\alpha^\bullet_i}) \cdot R^\star, \sum_{k=1}^{m^\bullet_{i,j}} r_{\gamma^\bullet_{i,j,k}} H_{0,l^\bullet_{i,j,k}})^r, i, j) = t_i \oplus H_2^\bullet([\prod_{k=1}^{m^\bullet_{i,j}} e(R_{\kappa^\bullet_{i,j,k}}, H_{0,l^\bullet_{i,j,k}})]^r, i, j)
\end{aligned}
$$

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

- ***CredGen-O***. Assume that adversary $\mathcal{A}^{\circ}$ makes a query on a tuple (I_{κ}, A^{pk_u}). Let $[A_{\iota}, H_{0,\iota}, \lambda_{\iota}]$ be the tuple from H_0^{list} such that $A_{\iota} = A^{pk_u}$, then algorithm $\mathcal{A}^{\bullet}$ responds as follows:
  1. If $\iota = l^{\bullet}_{i,j,1}$ and $\kappa \in \{\kappa^{\bullet}_{i,j,k}\}_{i,j,k}$, then report failure and terminate (event $\mathcal{E}_{\mathsf{cred}}$)
  2. If $\iota \neq l^{\bullet}_{i,j,1}$ and $\kappa \in \{\kappa^{\bullet}_{i,j,k}\}_{i,j,k}$, then return $(r_{\kappa}\lambda_{\iota}) \cdot R^{\star} = (r_{\kappa}s^{\star}) \cdot H_{0,\iota}$
  3. If $\kappa \in \{1,\ldots,N\} \setminus \{\kappa^{\bullet}_{i,j,k}\}_{i,j,k}$, then return $s_{\kappa} \cdot H_{0,\iota}$
- ***Decrypt-O***. Assume that adversary $\mathcal{A}^{\circ}$ makes an oracle query on a tuple $(C, Pol^{pk_{\mathrm{ch}}}, \{j_1,\ldots,j_m\})$. Then, algorithm $\mathcal{A}^{\bullet}$ responds as follows:
  1. If $Pol^{pk_{\mathrm{ch}}} \neq Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$ and $Pol^{pk_{\mathrm{ch}}}$ involves a condition $\langle I_{\kappa}, A^{pk_{\mathrm{ch}}}\rangle$ such that $\kappa \in \{\kappa^{\bullet}_{i,j,k}\}$ and $A^{pk_{\mathrm{ch}}} \in \{A_{l^{\bullet}_{i,j,1}}\}$, then report failure and terminate (event $\mathcal{E}_{\mathsf{dec}}$)
  2. If $Pol^{pk_{\mathrm{ch}}} \neq Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$ and $Pol^{pk_{\mathrm{ch}}}$ does not involve any condition $\langle I_{\kappa}, A\rangle$ such that $\kappa \in \{\kappa^{\bullet}_{i,j,k}\}$ and $A \in \{A_{l^{\bullet}_{i,j,1}}\}$, then do the following: (1) Run oracle *CredGen-O* multiple times until obtaining the qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_{\mathrm{ch}}})$, (2) Run algorithm *Decrypt* on input the tuple $(C, Pol^{pk_{\mathrm{ch}}}, pk_{\mathrm{ch}}, sk_{\mathrm{ch}}, \varsigma_{j_1,\ldots,j_m}(Pol^{pk_{\mathrm{ch}}}))$ and return the resulting output back to adversary $\mathcal{A}^{\circ}$
  3. If $Pol^{pk_{\mathrm{ch}}} = Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$, then do the following: let $C = (U, [[v_{i,j}]_{j=1}^{m^{\bullet}_i}]_{i=1}^{m^{\bullet}}, W)$, then compute the values $v^{\bullet}_i = v_{i,j_i} \oplus H_2^{\bullet}(j_i)\ \wr_{i=1}^{m^{\bullet}}$, and make a decryption query to the challenger on ciphertext $C^{\bullet} = (U, \hat{H}_2^{\bullet}(pk_{\mathrm{ch}}) \oplus (v^{\bullet}_1 \| \ldots \| v^{\bullet}_{m^{\bullet}}), W)$ and identifier $ID^{\bullet}$. 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 $\mathbb{G}_1$. Second, all the responses of algorithm $\mathcal{A}^{\bullet}$ to queries made by adversary $\mathcal{A}^{\circ}$ to oracles *CredGen-O* and *Decrypt-O* are consistent. Third, the ciphertext C_{ch} given to adversary $\mathcal{A}^{\circ}$ corresponds to the encryption according to $Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$ 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

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

From the simulation described above, we have

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

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

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

The total number of policies, distinct from policy $Pol_{\mathrm{ch}}^{pk_{\mathrm{ch}}}$, that may be specified by adversary $\mathcal{A}^{\bullet}$ during queries to oracle *Decrypt-O*, and that involve at least one of the conditions $\langle I_{\kappa}, A^{pk_{\mathrm{ch}}}\rangle$ such that $\kappa \in \{\kappa^{\bullet}_{i,j,k}\}$ and $A^{pk_{\mathrm{ch}}} \in \{A_{l^{\bullet}_{i,j,1}}\}$ could be upper bounded by the quantity

$$\Upsilon'(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) = \Upsilon(Nq_0, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) - \Upsilon(Nq_0 - (m_{\vee\wedge}m_{\vee})^2, m_{\vee\wedge}, m_{\vee}, m_{\wedge}) - 1$$

Then, the following statement holds

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

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

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

B Proof of Lemma 2

We construct an IND-CCA adversary $\mathcal{A}^{\bullet}$ that uses adversary $\mathcal{A}^{\circ}$ to mount an attack against the ElG-HybridPub scheme defined in [2]. 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 ElG-HybridPub system paramaters $PK^{\star} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, m^{\star}.n, H_1, H_2, H_3)$. Then, the challenger picks at random a private key $s^{\star} \in \mathbb{Z}_q^*$ and computes the corresponding public key $pk^{\star} = s^{\star} \cdot P$. Upon receiving $PK^{\star}$ and $pk^{\star}$, algorithm $\mathcal{A}^{\bullet}$ does the following:

1. Choose a hash function $\tilde{H}_2^{\bullet} : \{0,1\}^* \rightarrow \{0,1\}^n$
2. 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.

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*, *Phase-1*, *Challenge*, *Phase-2* and *Guess* which we describe below.

Setup. Algorithm $\mathcal{A}^{\bullet}$ does the following: (1) Let $\mathcal{P}^{\bullet} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_0, H_1, H_2^{\bullet}, H_3)$ be the system public parameters, where $H_0 : \{0,1\}^* \rightarrow \mathbb{G}_1$ is a randomly chosen hash function, the oracle $H_2^{\bullet}$ is controlled by algorithm $\mathcal{A}^{\bullet}$ and the tuple $(q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_1, H_3)$ is taken from $PK^{\star}$, (2) Define the the set of credential issuers $I = \{I_1, \ldots, I_N\}$ as follows: for $\kappa \in \{1, \ldots, N\}$, the public key of I_{κ} is $R_{\kappa} = s_{\kappa} \cdot P$ for some randomly chosen $s_{\kappa} \in \mathbb{Z}_q^*$, (3) Give the public parameters $\mathcal{P}^{\bullet}$, the public key $pk_{\mathrm{ch}} = pk^{\star}$ and the credential issuers' public and master keys $(R_{\kappa}, s_{\kappa})\,\wr_{\kappa=1}^{N}$ to adversary $\mathcal{A}^{\circ}$.

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

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

Challenge. Once adversary $\mathcal{A}^\circ$ decides that Phase-1 is over, it outputs two equal length messages M_0 and M_1 as well as a policy $Pol_{\text{ch}} = \wedge_{i=1}^{m}[\vee_{j=1}^{m_i}[\wedge_{k=1}^{m_{i,j}}\langle I_{\kappa_{i,j,k}}, A_{i,j,k}\rangle]]$ on which he wishes to be challenged. Adversary $\mathcal{A}^\bullet$ gives 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 EIG-HybridPub encryption of message M_b using the public key $pk^\star$. Upon receiving the challenger's response, compute $v_{i,j} = \Delta^\bullet(v^\star, i) \oplus \tilde{H}_2(\prod_{k=1}^{m_{i,j}} e(U, s_{\kappa_{i,j,k}} \cdot H_0(A_{i,j,k}))\|i\|j)$ $\wr_{j=1}^{m_i^\bullet} \wr_{i=1}^{m^\bullet}$, then forward the ciphertext $C_{\text{ch}} = (U, [[v_{i,j}]_{j=1}^{m_i^\bullet}]_{i=1}^{m^\bullet}, W)$ to adversary $\mathcal{A}^\circ$.
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 $U = r \cdot P$ for $r = H_1(M_b\|t)$, $W = M_b \oplus H_3(t)$ (for some randomly chosen $t \in \{0,1\}^{m^\bullet.n}$), and $v^\star = t \oplus H_2(r \cdot pk^\star)$. Let $t_i = \Delta^\bullet(t, i)$, then the following holds

$$\begin{aligned} v_{i,j} &= \Delta^\bullet(t \oplus H_2(r \cdot pk^\star), i) \oplus \tilde{H}_2(\prod_{k=1}^{m_{i,j}} e(r \cdot P, s_{\kappa_{i,j,k}} \cdot H_0(A_{i,j,k}))\|i\|j) \\ &= t_i \oplus \Delta^\bullet(H_2(r \cdot pk^\star), i) \oplus \tilde{H}_2(\prod_{k=1}^{m_{i,j}} e(s_{\kappa_{i,j,k}} \cdot P, H_0(A_{i,j,k}))^r\|i\|j) = t_i \oplus H_2^\bullet(r \cdot pk^\star\|\pi_{i,j}^r\|i\|j) \end{aligned}$$

Phase-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 *Decrypt-O* to which adversary $\mathcal{A}^\circ$ makes queries during *Phase-1* and *Phase-2* are described below.

- ***CredGen-O***. Since adversary $\mathcal{A}^\circ$ has access to the oracle H_0 and the master keys of the different credential issuers, it does not need to make queries to this oracle.
- ***Decrypt-O***. Assume that adversary $\mathcal{A}^\circ$ makes an oracle query on a tuple $(C, Pol^{pk_{\text{ch}}}, \{j_1, \ldots, j_m\})$. Then, algorithm $\mathcal{A}^\bullet$ responds as follows: let $C = (U, [[v_{i,j}]_{j=1}^{m_i^\bullet}]_{i=1}^{m^\bullet}, W)$, then compute the values $v_i^\bullet = v_{i,j_i} \oplus \tilde{H}_2(\prod_{k=1}^{m_{i,j_i}} e(U, s_{\kappa_{i,j_i,k}} \cdot H_0(A_{i,j_i,k}))\|i\|j_i)$, 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 $\mathcal{A}^\circ$

In the simulation described above, 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 $\mathbb{G}_1$. Second, all the responses of algorithm $\mathcal{A}^\bullet$ to queries made by adversary $\mathcal{A}^\circ$ to oracles *CredGen-O* and *Decrypt-O* are consistent. Third, the ciphertext C_{ch} given to adversary $\mathcal{A}^\circ$ corresponds to the encryption according to $Pol_{\text{ch}}^{pk_{\text{ch}}}$ of M_b for some random $b \in \{0,1\}$. Therefore, $\text{Adv}_{\mathcal{A}^\bullet} \geq \varepsilon$.