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

Research Report[a] N^o 169 — RR-06-169

An Application of Policy-Based Signature: Proof-Carrying Proxy Certificates

Walid Bagga and Stefano Crosta and Refik Molva

April, 2006

[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

An Application of Policy-Based Signature: Proof-Carrying Proxy Certificates

Abstract. The term proxy certificate is used to describe a certificate that is issued by an end user for the purpose of delegating responsibility to another user so that the latter can perform certain actions on behalf of the former. Such certificates have been suggested for use in a number of applications, particularly in distributed computing environments where delegation of rights is common. In this paper, we present a new concept called *proof-carrying proxy certificates*. Our approach allows to combine the verification of the validity of the proxy certificate and the authorization decision making in an elegant way that enhances the privacy of the end user. In contrast with standard proxy certificates that are generated using standard (public-key) signature schemes, the proposed certificates are generated using a signature scheme for which the validity of a generated signature proves the compliance of the signer with a credential-based policy. We present a concrete realization of our approach using bilinear pairings over elliptic curves and we prove its security under adapted attack models.

Keywords: Proxy Certificates, Credentials, Authorization, Bilinear Pairings, Data Minimization

1 Introduction

The concept of proxy certificates, first formalized in [15], allows an end user to delegate some responsibility to another user, called agent, so that the latter can perform certain actions on behalf of the former. A proxy certificate is a certificate that, in contrast with the public-key certificates issued by trusted certification authorities (such as X.509 certificates), is generated by an end user. It represents the signature of the end user on a message that typically contains the identity of the end user himself, the public key of the agent and a set of statements defining the terms of the delegation. It allows the agent to authenticate with other users as if he was the end user when performing the delegated actions. Proxy certification has been suggested for use in a number of applications particularly in distributed computing environments where delegation of rights is quite common. Examples include grid computing [5], mobile agents for e-commerce [7], and mobile communication [6]. More recently, an X.509 certificate profile for proxy certificates was proposed in [18].

Whenever an agent wants to perform an action on behalf of an end user, he must prove that he is authorized by the end user to perform the action on his behalf. This is achieved by providing a valid proxy certificate and proving the possession of the private key corresponding to the agent's public key specified by the certificate. Furthermore, the agent has to prove that the end user is compliant with the authorization policy associated to the action he wants to perform. An increasingly popular approach for authorization in large-scale open environments like the Internet consists in using policies fulfilled by digital credentials. Basically, a digital credential is composed of a set of statements about certain user and the signature of this set by a trusted entity (called credential issuer). In this context, a commonly taken approach consists in that the agent provides a set of end user's credentials fulfilling the authorization policy (called a qualified set of credentials for the policy). The entity that is in charge of making the authorization decision is called the verifier. On one hand, the verifier has to check the validity of each of the received credentials. On the other hand, he has to check that the received set of credentials fulfills the authorization policy associated to the requested action.

The standard approach is not satisfactory for three reasons: first, verifying the validity of the proxy certificate and the validity of the different credentials separately is a burden for the verifier. Second, we believe that managing the end user's credentials and proving his compliance with an authorization policy should not be the role of the agent. Third, proving the compliance with a credential-based policy through the disclosure of a qualified set of credentials is not optimal from a privacy point of view. More precisely, it is not compliant with the privacy principle of data minimization (called the data quality principle in OECD guidelines [9]) that states that only strictly necessary information should be collected for a given purpose. For instance, assume that the authorization policy requires the possession of at least one credential belonging to a set of multiple credentials. Then, according to the data minimization principle, the verifier should not know more than the fact that the end user is compliant with the policy. In other words, the verifier should not know which specific credential fulfilling the authorization policy is held by the end user.

In this paper, we introduce a novel form of proxy certificates called *proof-carrying proxy certificates*. In contrast with standard proxy certificates that are generated using standard (public-key) signature schemes, the proposed certificates are generated using a signature scheme for which the validity of a generated signature proves the compliance of the signer with a credential-based policy. Using this special form of proxy certificates, the end user does not disclose any of his credentials. He uses them to generate a proof of compliance with the verifier's authorization policy. Besides, the agent does not have to deal with the end user's credentials. He just provides his proof-carrying proxy certificate (in addition to proving the possession of the private key corresponding to the agent's public key specified by the certificate). Finally, the verifier will just need to verify the validity of the received proxy certificate with respect to his policy i.e. the verification of the validity of the proxy certificate and the authorization decision making are performed in a logically single step.

The signature scheme used for the generation of proof-carrying proxy certificates should be unforgeable as for standard signature schemes. Furthermore, the scheme has to fulfill a privacy property called credential ambiguity in order to fulfill the data minimization principle i.e. the validity of a the signature on the proof-carrying proxy certificate proves that the end user is compliant with the authorization policy. However, if multiple qualified sets of credentials can fulfill the policy, the verifier should not know which specific one is held by the end user. In the following, an application scenario is described as an illustration of our approach.

Application Scenario. Consider the following scenario: a researcher (end user) wants to perform some operations on various hosts on a scientific computation oriented grid environment. The operations can be executed independently, can depend on each other, or can be executed only at specific periods of time. From his laptop the researcher wants to submit a number of requests to the destination hosts and have the operations executed while he is doing other things including being offline.

For each request, an authenticated connection needs to be established with the corresponding destination host. An authorization policy is associated to the operations and the researcher has to prove his compliance with the policy in order for the operations to be authorized to be executed. The researcher delegates the management of the different operations to one or more agents.

Currently, authorization in grid environments is identity-based. The researcher whose public/private key pair is denoted (pk_u, sk_u) holds an X.509 certificate binding his global identity to his public key. In order to make the agent act on his behalf, he generates for the agent a random pair of keys denoted (pk_a, sk_a). Then, he issues an X.509 proxy certificate [18] associated to the generated key pair. The certificate contains in addition to the agent's public key pk_a, a set of statements indicating the valid operations that the agent is allowed to perform on behalf of the researcher, as well as a restricted validity period. The authentication of the agent is therefore based on its key pair, the proxy certificate generated by the researcher and the public-key certificate of the researcher. Authorization to perform a specific task is based on the identity of the researcher (taken from his X.509 certificate) as well as on the statements within the proxy certificate.

As explained in [5], an identity-based approach to authorization and authentication for large grids "will not provide the scalability, flexibility, and ease of management that a large grid needs to control access to its sensitive resources", while a property-based approach where properties are carried by digital credentials is more appropriate. In scientific grids for instance, properties may include whether the requesting agent is acting on behalf of a professor, a student or an administrator; whether the agent is acting on behalf of a member of a particular research project whose membership list is not maintained locally; whether the agent is acting on behalf of a researcher from academy or industry; etc.

In the credential-based approach, the agent needs to prove that its owner (the researcher) is compliant with a specific credential-based authorization policy in order for the operations to be executed. Using standard credential systems such as X.509 attribute certificates, the agent needs to have access to the credentials of its owner to provide the necessary authorization arguments. For example, assume that a policy requires the researcher to be either a research staff member of company X or company Y. Suppose that the researcher is employed by company X, therefore he has been issued a credential $cred_X^u$ (associated to his public key pk_u). In addition to the proxy certificate, the researcher gives to the agent the credential $cred_X^u$. During authentication and authorization phase, the agent submits in addition to its proxy certificate, the researcher's credential $cred_X^u$. The remote host where the operation needs to be executed does the following: (1) check the validity of the proxy certificate using the public key pk_u, (2) check the validity of $cred_X^u$ using the public key of the 'trusted' credential issuer, (3) check whether the provided credential fulfills the authorization policy for the requested operations. If all the validity checks are successful, the task is executed. Otherwise, an error message is returned.

Using proof-carrying proxy certificates allows to combine the verification of the validity of the proxy certificate and the authorization decision making in a way that improves the privacy of the researcher. In fact, instead of using a standard signature scheme, the researcher generates the agent's proxy certificate by running an advanced signature algorithm on input of his private key

sk_u, his credential $cred_X^u$ and the credential-based policy '$cred_X^u$ or $cred_Y^u$'. The new proxy certificate carries in addition to delegation rights, the authorization arguments necessary for the execution of the operations. Hence, instead of performing three validity checks, the remote host needs just to verify the validity of the proxy certificate with respect to the policy '$cred_X^u$ or $cred_Y^u$' using the researcher's public key pk_u. Furthermore, thanks to the credential ambiguity property, the remote host will not know whether the agent is acting on behalf of a company X or company Y.

Contributions and Organization of the Paper. In this paper, we present the concept of proof-carrying proxy certificates that allows to combine the verification of the validity of the proxy certificate and the authorization decision making in a way that enhances the privacy of the end user. After discussing the related work in Section 2, we provide a comprehensive overview of the proof-carrying proxy certification mechanism in Section 3. In Section 4, we provide precise definitions for the algorithms specifying a proof-carrying proxy certification scheme. Then, we define the related security models, namely unforgeability and credential ambiguity. In Section 5, we describe a provably secure construction of proof-carrying proxy certification scheme based on bilinear pairings over elliptic curves. In Section 6, we summarize the paper and discuss current and future research work.

2 Related Work

The intuition behind the concept of proof-carrying proxy certificates comes originally from proof-carrying codes [14]. The latter is a technique that can be used for safe execution of untrusted code. In a typical scenario, a code receiver establishes a set of safety rules that guarantee safe behavior of programs, and the code producer creates a formal safety proof that proves, for the untrusted code, adherence to the safety rules. Then, the receiver is able to use a proof validator to check that the proof is valid and hence the untrusted code is safe to execute. By analogy with proof-carrying codes, a proof-carrying authentication mechanism based on higher-order logic was presented in [1]: the client desiring access must construct a proof using his attribute certificates, and the server will simply check the validity of the proof. The logic-based approach leads to a simple and efficient solution that integrates different authentication frameworks including X.509 and SPKI/SDSI. However, it cannot be used in the context of proof-carrying proxy certification because it does not provide a signature scheme fulfilling the required properties.

Providing a privacy preserving proof of compliance with a credential-based policy is a problem that has been studied in recent literature. In [2], the authors exploit cryptographic zero-knowledge proofs to allow requesting users to prove their adherence with a credential-based policy. The proposed solution provides better privacy guarantees than our concrete implementation of proof-carrying proxy certificates as the users may prove their compliance while preserving their anonymity. However, as the described protocol requires interaction between the credentials holder (end user) and the verifier, it can not be directly used to implement proof-carrying proxy certificates. An interesting line for future research would be to exploit the Fiat-Shamir heuristic [8] to transform their interactive protocols into a signature scheme that could be used to implement proof-carrying proxy certificates.

The concept of self-certified signatures presented in [12] shares with proof-carrying proxy certificates the idea of combining signature's validity verification with certification information verification: the signer (end user) first generates a temporary signing key (analog to the agent's private key) using his long-term signing key and his public-key certification information together. Then, he signs a message and certification information using this temporary signing key. In the verification stage both the signature on the message and certification are checked together. Self-certified signature was extended to multi-certification signature in which multiple certificates are verified together with the signature. The multi-certification signature scheme described in [12] could be used to construct proof-carrying proxy certificates for which policies are restricted to conjunctions of credentials. However, they cannot support disjunctions of credentials while respecting the credential ambiguity property. Thus, the signature scheme used in proof-carrying proxy certification could be seen as a generalization of self-certified signatures that supports both disjunctive and conjunctive authorization structures.

Our pairing-based signature scheme for proof-carrying proxy certificates is based on the policy-based signature scheme proposed in [3]. The latter allows to generate a signature on a message so that the signature is valid if and only if the signer is compliant with a credential-based policy written in standard normal form. However, it cannot be used to implement proof-carrying proxy certificates as it suffers from collusion attacks. In fact, in addition to the legitimate signer, any collusion of credential issuers or end users who are able to collect a qualified set of credentials for the policy according to which the message is signed can generate a valid signature. Besides, the scheme is not satisfactory as it is not supported by formal security arguments. In this paper, we propose a scheme that solves the collusion problem and provides a formal security analysis based on reductionist proofs, thus fulfilling the security requirements of proof-carrying proxy certificates.

3 Proof-Carrying Proxy Certification

In this section, we provide a general description of our approach as well as the notations used along the paper. We define the different components of a proof-carrying proxy certification scheme, including our policy model. Then, we describe how the proof-carrying proxy certificates are created and used.

3.1 Setting the Context

The setting for proof-carrying proxy certification comprises four types of players: end users, credential issuers, agents and verifiers (service providers). 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.

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 validity of the assertion. 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.

Each service provider defines an authorization policy for each action on a sensitive resource he controls. 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 the end user 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 [17]. 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 $m_{i,j} = 1$, for all 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 $\{j_i \in \{1, \ldots, m_i\}\}_{i=1}^{m}$. Then, $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$ is a qualified set of credentials for Pol^{pk_u}.

3.2 Creating and Using Proof-Carrying Proxy Certificates

When an end user wants to interact with a service provider (verifier) through an agent, he first generates a pair of keys (pk_a, sk_a) for the agent. Then, he specifies the content of the proxy certificate - a message, denoted M, containing the end user's public key pk_u, the public key of the agent pk_a and the delegation constraints. Finally, the end user generates a signature on the content of the proxy certificate using a dedicated signature algorithm. The latter takes as input the message to be signed, the private key of the end user sk_u, the policy of the service provider Pol^{pk_u} with respect to the end user's public key pk_u, and a qualified set of credentials for the policy $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$.

When the agent decides to interact with the verifier, he provides his proof-carrying proxy certificate along with a proof of possession of the private key sk_a corresponding to the public key pk_u contained in the proxy certificate. The verifier first checks the delegation constraints specified by the proxy certificate to be sure that the agent is allowed by the end user to perform the requested action on his behalf. Then, he checks the validity of the signature on the content of the proxy certificate using the adequate verification algorithm. This algorithm takes as input the proof-carrying proxy certificate, the end user's public key pk_u, and the authorization policy Pol^{pk_u}. At the end, the verifier obtains a proof that the agent whose public key is pk_a is allowed by an end user whose public key is pk_u to perform the action on his behalf and that the end user is compliant with the authorization policy specified by the verifier.

The signature and verification algorithms used for the creation and verification of proof-carrying proxy certificates must fulfill two security requirements:

- Unforgeability: the signature on a proof-carrying proxy certificate must not be valid with respect to policy Pol^{pk_u} if the signer does not use the private key sk_u or a qualified set of credentials for policy Pol^{pk_u}. In other words, the agent cannot obtain a valid proof-carrying proxy certificate with respect to policy Pol^{pk_u} from a user that does not have access to the private key sk_u, and the end user cannot generate a valid proof-carrying proxy certificate with respect to policy Pol^{pk_u} if he does not have access to a qualified set of credentials for the policy.
- Credential ambiguity: in the case where there exists multiple qualified sets of credentials for policy Pol^{pk_u}, a valid proxy-carrying proxy certificate must not reveal which specific set of credentials has been used to generate the certificate.

4 Definitions

Following the functional description provided in Section 3, we give in this section precise definitions for the algorithms used during the proof-carrying proxy certification process. In addition, we formally define the corresponding security models.

4.1 Algorithms

A proof-carrying proxy certification scheme (in short PCPC) is specified by six algorithms: *System-Setup*, *Issuer-Setup*, *User-Setup*, *CredGen*, *Sign* and *Verify*.

System-Setup. On input of a security parameter k, this algorithm generates the system parameters $\mathcal{P}$ including the different spaces, groups and public functions that will be referenced by subsequent algorithms.

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 $I_\kappa \in \mathcal{I}$ and an assertion $A^{pk_u} \in \{0,1\}^*$, this algorithm generates the credential $\varsigma(R_\kappa, A^{pk_u})$ using the master key s_κ associated to R_κ.

Sign. On input of a message M, 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 a signature σ.

Verify. On input of a message M, a signature σ, a public key pk_u and a policy Pol^{pk_u}, this algorithm returns $\top$ (for *true*) if σ is a valid signature on M according to policy Pol^{pk_u}. Otherwise, it returns $\perp$ (for *false*).

The algorithms described above have to satisfy the standard consistency constraint i.e.

$$\sigma = Sign(M, pk_u, sk_u, Pol^{pk_u}, \varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})) \Rightarrow Verify(M, \sigma, pk_u, Pol^{pk_u}) = \top$$

4.2 Security Models

A PCPC scheme has to fulfill the security requirement of unforgeability and the privacy requirement of credential ambiguity.

Unforgeability. The standard acceptable notion of security for standard signature schemes is existential unforgeability against chosen message attacks [10]. Therefore, we require the same security notion for proof-carrying proxy certification schemes. The definition of existential unforgeability should naturally be adapted to the advanced form of signature used by proof-carrying proxy certificates.

Existential unforgeability for PCPC schemes is defined in terms of an interactive game, played between a challenger and an adversary. The game consists of three stages: *Setup*, *Queries* and *Forge* 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 $\mathcal{P}$, (2) Run algorithm *Issuer-Setup* once or multiple times to obtain a set of credential issuers $\mathcal{I} = \{I_1, \ldots, I_N\}$, (3) Run algorithm *User-Setup* to obtain a public/private key pair $(pk_\mathrm{f}, sk_\mathrm{f})$, (4) Give to the adversary the parameters $\mathcal{P}$, the public key pk_f and the public keys of the different credential issuers included in $\mathcal{I}$.
– ***Queries***. The adversary performs adaptively a polynomial number of oracle queries which we define below. By ”adaptively”, we mean that each query may depend on the challenger’s replies to the previously performed queries.
– ***Forge***. Once the adversary decides that *Queries* is over, it outputs a message M_f, a policy $Pol_\mathrm{f}^{pk_\mathrm{f}}$, a signature σ_f, and wins the game if $\mathsf{Verify}(M_\mathrm{f}, \sigma_\mathrm{f}, pk_\mathrm{f}, Pol_\mathrm{f}^{pk_\mathrm{f}}) = \top$.

During the *Queries* stage, 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 signature oracle denoted ***Sign-O***. While the oracles are executed by the challenger, their input is specified by the adversary. The oracles are defined below:

- ***CredGen-O***. On input of a credential issuer $I_\kappa \in \mathcal{I}$ and an assertion $A^{pk_u} \in \{0,1\}^*$ (associated to a key pair (pk_u, sk_u) chosen by the adversary), run algorithm *CredGen* on input of the tuple (I_κ, A^{pk_u}) and return the resulting credential $\varsigma(R_\kappa, A^{pk_u})$.
- ***Sign-O***. On input of a message M and a policy Pol^{pk_f}, first run algorithm *CredGen* once or multiple times to obtain a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_\mathrm{f}})$ for Pol^{pk_f}, then run algorithm *Sign* on input of $(M, pk_\mathrm{f}, sk_\mathrm{f}, Pol^{pk_\mathrm{f}}, \varsigma_{j_1,\ldots,j_m}(Pol^{pk_\mathrm{f}}))$ (for some $j_i \in \{1,\ldots,m_i\}$ for $i = 1,\ldots,m_i$) and return the resulting output.

The oracle queries made by the adversary during *Queries* are subject to some restrictions depending on the type of adversary. In fact, we distinguish two types of attackers:

- Insider: the adversary is given, in addition to the parameters provided by the challenger during *Setup*, the private key sk_f. An adversary of this type is not allowed to obtain (through queries to oracle *CredGen-O*) a qualified set of credentials for the forgery policy $Pol_\mathrm{f}^{pk_\mathrm{f}}$. This type of attackers corresponds to entities that are not compliant with a policy and that try to generate a valid signature w.r.t the policy.
- Outsider: the adversary is given, in addition to the parameters provided by the challenger during *Setup*, the master keys of the different credential issuers included in $\mathcal{I}$. An adversary of this type does not have access to the private key sk_f and do not need to perform queries to oracle *CredGen-O*. This type of attackers corresponds to entities that might have access to a qualified set of credentials for the policy but do not have access to the corresponding public key.

Obviously, an adversary, be it insider or outsider, is not allowed to perform a query to oracle *Sign-O* on the tuple $(M_\mathrm{f}, Pol_\mathrm{f}^{pk_\mathrm{f}})$.

The game described above is denoted $\mathsf{EUF\text{-}PCPC\text{-}CMA}^\mathsf{X}$, where $\mathsf{X} = \mathsf{I}$ for insider adversaries and $\mathsf{X} = \mathsf{O}$ for outsider adversaries. A formal definition of existential unforgeability against chosen message attacks for PCPC schemes is given below. As usual, a real function g is said to be negligible if $g(k) \le \frac{1}{f(k)}$ for any polynomial f.

Definition 1. *The advantage of an adversary $\mathcal{A}^X$ in the* $\mathsf{EUF\text{-}PCPC\text{-}CMA}^X$ *game is defined to be the quantity $Adv_{\mathcal{A}^X} = Pr[\mathcal{A}^X\ wins]$. A* PCPC *scheme is* $\mathsf{EUF\text{-}PCPC\text{-}CMA}^X$ *secure if no probabilistic polynomial time adversary has a non-negligible advantage in the* $\mathsf{EUF\text{-}PCPC\text{-}CMA}^X$ *game.*

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

- ***Setup***. On input of a security parameter k, the challenger does the following: (1) Run algorithm *Setup* to obtain the system public parameters $\mathcal{P}$, (2) Run algorithm *Issuer-Setup* once or multiple times to obtain a set of credential issuers $\mathcal{I} = \{I_1, \dots, I_N\}$, (3) Give to the adversary the parameters $\mathcal{P}$ as well as the public and master keys of the different credential issuers included in $\mathcal{I}$.
- ***Challenge***. The adversary chooses a message M_{ch}, a pair of keys $(pk_{\text{ch}}, sk_{\text{ch}})$ and a policy $Pol_{\text{ch}}^{pk_{\text{ch}}}$ on which he wishes to be challenged. The challenger does the following: (1) For $i = 1, \dots, m$, pick at random $j_i^{\text{ch}} \in \{1, \dots, m_i\}$, (2) Run algorithm *CredGen* m times to obtain the qualified set of credentials $\varsigma_{j_1^{\text{ch}}, \dots, j_m^{\text{ch}}}(Pol_{\text{ch}}^{pk_{\text{ch}}})$, (3) Run algorithm *Sign* on input the tuple $(M_{\text{ch}}, pk_{\text{ch}}, sk_{\text{ch}}, Pol_{\text{ch}}, \varsigma_{j_1^{\text{ch}}, \dots, j_m^{\text{ch}}}(Pol_{\text{ch}}^{pk_{\text{ch}}}))$ and return the resulting output to the adversary.
- ***Guess***. The adversary outputs a tuple $(j_1, \dots, j_m)$, and wins the game if the equality $(j_1^{\text{ch}}, \dots, j_m^{\text{ch}}) = (j_1, \dots, j_m)$ holds.

Definition 2. *The advantage of an adversary $\mathcal{A}$ in the* CrA-PCPC-CMA *game is defined to be the quantity $Adv_{\mathcal{A}} = \mathsf{Max}_i\{|Pr[j_i = j_i^{ch}] - \frac{1}{m_i}|\}$, where the parameters m_i are those defined by the challenge policy $Pol_{ch}^{pk_{ch}}$. A* PCPC *scheme is* CrA-PCPC-CMA *secure if no probabilistic polynomial time adversary has a non-negligible advantage in the* CrA-PCPC-CMA *game.*

5 Concrete Implementation

In this section, we describe a concrete implementation of proof-based proxy certificates. Our implementation is based on bilinear pairings over elliptic curves. Our scheme owes much to the work on pairing-based signature and ring signatures presented in [13, 19, 20]. After describing our concrete algorithms, we analyze their consistency and efficiency. Then, we prove their security in the random oracle model.

5.1 Description

Before describing the algorithms defining our PCPC 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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ 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 following mathematical problem are hard to solve:

- Computational Diffie-Hellman Problem (CDHP): given a tuple $(P, a \cdot P, b \cdot P)$ for randomly chosen $a, b \in \mathbb{Z}_q^*$, compute the value $ab \cdot P$.
- $(k+1)$-Exponent Problem ($k+1$EP): given the tuple $(P, a \cdot P, a^2 \cdot P, \dots, a^k \cdot P)$ for $a \in \mathbb{Z}_q^*$, compute $a^{k+1} \cdot P$.

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 $Q,Q' \in \mathbb{G}_1$. $\diamond$

Our PCPC scheme consists of the algorithms described below.

System-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. Define three hash functions: $H_0 : \{0,1\}^* \rightarrow \mathbb{G}_1$, $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 : \mathbb{G}_1 \rightarrow \mathbb{Z}_q^*$
3. Let $\mathcal{P} = (q,\mathbb{G}_1,\mathbb{G}_2,e,P,H_0,H_1,H_2)$.

Issuer-Setup. Let $\mathcal{I} = \{I_1,\ldots,I_N\}$ be a set of credential issuers. Each credential issuer $I_\kappa \in \mathcal{I}$ 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 the public key R_κ of issuer $I_\kappa \in \mathcal{I}$ and assertion $A^{pk_u} \in \{0,1\}^*$, this algorithm outputs $\varsigma(R_\kappa, A^{pk_u}) = s_\kappa \cdot H_0(A^{pk_u})$.

Sign. On input of a message M, a pair of keys (pk_u, sk_u), a policy Pol^{pk_u} and a qualified set of credentials $\varsigma_{j_1,\ldots,j_m}(Pol^{pk_u})$, do the following:

```
1. For $i = 1,\ldots,m$, do the following:
   (a) Pick at random $Y_i \in \mathbb{G}_1$, then compute $x_{i,j_i+1} = e(P,Y_i)$
   (b) For $l = j_i+1,\ldots,m_i,1,\ldots,j_i-1 \bmod (m_i+1)$, do the following:
       i. Compute $\tau_{i,l} = \prod_{k=1}^{m_{i,l}} e(R_{\kappa_{i,l,k}}, H_0(A_{i,l,k}^{pk_u}))$
       ii. Pick at random $Y_{i,l} \in \mathbb{G}_1$, then compute $x_{i,l+1} = e(P,Y_{i,l}) * \tau_{i,l}^{H_1(M\|x_{i,l}\|m\|i\|l)}$
   (c) Compute $Y_{i,j_i} = Y_i - H_1(M\|x_{i,j_i}\|m\|i\|j_i) \cdot (\sum_{k=1}^{m_{i,j_i}} \varsigma(R_{\kappa_{i,j_i,k}}, A_{i,j_i,k}^{pk_u}))$
2. Compute $Y = \sum_{i=1}^{m}\sum_{j=1}^{m_i} Y_{i,j}$, then compute $Z = (sk_u + H_2(Y))^{-1} \cdot P$
3. Return $\sigma = ([[x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y, Z)$
```

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

```
1. Compute $\tau_{i,j} = \prod_{k=1}^{m_{i,j}} e(R_{\kappa_{i,j,k}}, H_0(A_{i,j,k}^{pk_u}))$ (for $j = 1,\ldots,m_i$ and $i = 1,\ldots,m$)
2. Compute $\alpha_0 = e(pk_u + H_2(Y)\cdot P, Z)$
3. Compute $\alpha_1 = \prod_{i=1}^{m}[\prod_{j=1}^{m_i} x_{i,j}]$ and $\alpha_2 = e(P,Y) * \prod_{i=1}^{m}\prod_{j=1}^{m_i} \tau_{i,j}^{H_1(M\|x_{i,j}\|m\|j\|i)}$
4. If $\alpha_0 = e(P,P)$ and $\alpha_1 = \alpha_2$, then return $\top$, otherwise return $\perp$
```

The intuition behind our signature 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 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

$\{\varsigma(R_{\kappa_{i,j,k}}, A^{pk_u}_{i,j,k})\}_{k=1}^{m_{i,j}}$. Thus, the generated signature corresponds to a set of ring signatures which validity can be checked using the global 'glue' value Y. The latter can be computed only by a user having access to a qualified set of credentials for policy Pol^{pk_u}. The element Z represents the [20] short signature on Y using the private key sk_u. Therefore, σ proves that the entity whose public key is pk_u is compliant with policy Pol^{pk_u}. Note that we can use any standard signature scheme to generate the value Z.

5.2 Consistency and Efficiency

Our *PCPC* scheme satisfies the standard consistency constraint thanks to the following statements:

$$\alpha_0 = e(pk_u + H_2(Y)\cdot P, Z) = e((sk_u + H_2(Y))\cdot P, (sk_u + H_2(Y))^{-1}\cdot P) = e(P,P) \quad (1)$$

$$\tau_{i,j}^{H_1(M\|x_{i,j}\|m\|i\|j)} = x_{i,j+1} * e(P, Y_{i,j})^{-1} (\text{where} x_{i,m_i+1} = x_{i,1}) \quad (2)$$

$$\begin{aligned}
\alpha_2 &= \lambda * \prod_{i=1}^{m}[\prod_{j=1}^{m_i} \tau_{i,j}^{H_1(M\|x_{i,j}\|m\|i\|j)}] \quad (\text{where } \lambda = e(P,Y)) \\
&= \lambda * \prod_{i=1}^{m}[\prod_{j=1}^{m_i-1} x_{i,j+1} * e(P,Y_{i,j})^{-1} * x_{i,1} * e(P, Y_{i,m_i})^{-1}] \\
&= \lambda * \prod_{i=1}^{m}[\prod_{j=1}^{m_i} x_{i,j} * \prod_{j=1}^{m_i} e(P,Y_{i,j})^{-1}] \\
&= \lambda * [\prod_{i=1}^{m}\prod_{j=1}^{m_i} x_{i,j}] * [e(P, \sum_{i=1}^{m}\sum_{j=1}^{m_i} Y_{i,j})]^{-1} = \lambda * \alpha_1 * \lambda^{-1} \quad (3)
\end{aligned}$$

The essential operation in pairing-based cryptography is pairing computations. Our signature algorithm requires a total of $\sum_{i=1}^{m} m_i + \sum_{i=1}^{m}\sum_{j\neq j_i} m_{i,j}$ pairing computations. Note that the values $\tau_{i,l}$ does not depend on the signed message M. Thus, they can be pre-computed by the end user, cached and used in subsequent signatures involving the corresponding credential-based conditions i.e. $\langle R_{\kappa_{i,l,k}}, A^{pk_u}_{i,l,k}\rangle$. On the other hand, our verification algorithm requires a total of $3 + \sum_{i=1}^{m}\sum_{j=1}^{m_i} m_{i,j}$ pairing computations. Although pairing computations could be optimized as explained in [4], the performance of our signature and verification algorithms still need to be improved. This is the main focus of our current research work.

Let l_i denote the bit-length of the bilinear representation of an element of group $\mathbb{G}_i$ ($i = 1,2$). Then, the bit-length of a signature produced by our PCPC scheme is equal to $(\sum_{i=1}^{m} m_i).l_2 + 2.l_1$. Note that the signature's length does not depend on the values $m_{i,j}$.

5.3 Security

In the following, we provide the security results related to our PCPC scheme.

Notation. Given the notation used in Section 3, 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* PCPC *scheme is* EUF-PCPC-CMA^I *secure in the random oracle model under the assumption that CDHP is hard. In fact, let $\mathcal{A}^\circ$ be an* EUF-PCPC-CMA^I *adversary with advantage $Adv_{\mathcal{A}^\circ} \geq \varepsilon$ when attacking our* PCPC *scheme. 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 Sign-O, q_0 queries to oracle H_0 and q_1 queries to oracle H_1. Then, there exists an adversary $\mathcal{A}^\bullet$ the advantage of which, when attacking CDHP, is such that*

$$Adv_{\mathcal{A}^\bullet} \geq 9/(100q_0^{m_{\vee\wedge}m_{\vee}} \textstyle\sum_{l=1}^{m_{\vee}} l!\binom{m_{\vee}}{l})$$

Its running time is $t_{\mathcal{A}^\bullet} \leq (32q_1+4)t_{\mathcal{A}^\circ}/\varepsilon$, for $q \geq Max\{2m_{\vee\wedge}m_{\vee}, 2m_{\vee\wedge}q_sq_1\}$ and $\varepsilon \leq 32(q_1+1-m_{\vee\wedge}m_{\vee})/q$.

Proof. Proof of Theorem 1 follows the method described in [11], which is based on the oracle replay technique [16]. 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 A. Note that our security reduction does not depend on the parameter $m_{\wedge}$. On the other hand, it depends exponentially on the parameters $m_{\vee\wedge}$ and $m_{\vee}$ which needs further improvement. Finally, note that the ID-based ring signature presented in [20] is not supported by any security arguments. Our proof could be easily adapted to realize the missing proofs. In fact, the ID-based ring signature of [20] is almost similar to our signature algorithm applied in the particular case where the policies are such that $m_{\vee\wedge} = m_{\wedge} = 1$. $\square$

Theorem 2. *Our* PCPC *scheme is* EUF-PCPC-CMA^O *secure in the random oracle model under the assumption that $k+1$EP is hard.*

Proof. The security of our scheme PCPC in the EUF-PCPC-CMA^O game is equivalent to the security of the short signature scheme presented in [20]. In fact, the outsider adversary succeeds in forging a proof-carrying proxy certification if and only if it succeeds in generating a valid Z corresponding to a valid $([[x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y)$ associated to the pair of keys $(pk_{\mathrm{f}}, sk_{\mathrm{f}})$. As the adversary has access to the master keys of the different credential issuers, its is able to generate a valid tuple $([[x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y)$ corresponding to any policy associated to pk_{f}. Therefore, the adversary needs to be able to generate a [20] short signature on Y using the protected private key sk_{f}. The short signature of [20] is proved to be secure in the random oracle model under the assumption that the k+1EP problem is hard. $\square$

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

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

1. $x^{\mathrm{ch}}_{i,j} = e(P, Y_{i,j-1}) * \tau_{i,j-1}^{H_1(M_{\mathrm{ch}} \| x^{\mathrm{ch}}_{i,j-1} \| m \| i \| j-1)}$ for $j \neq j_i^{\mathrm{ch}} + 1$ and $x^{\mathrm{ch}}_{i,j_i^{\mathrm{ch}}+1} = e(P, Y_i)$
2. $Y^{\mathrm{ch}} = \sum_{i=1}^{m} [\sum_{j \neq j^{\mathrm{ch}}} Y_{i,j} + Y_i - H_1(M_{\mathrm{ch}} \| x^{\mathrm{ch}}_{i,j_i^{\mathrm{ch}}} \| m \| i \| j_i^{\mathrm{ch}}) \cdot (\sum_{k=1}^{m_{i,j_i^{\mathrm{ch}}}} \varsigma(R_{\kappa_{i,j_i^{\mathrm{ch}},k}}, A_{i,j_i^{\mathrm{ch}},k}))]$

Since Y_i and $Y_{i,j-1}$ are chosen at random from $\mathbb{G}_1$, and H_1 is assumed to be a random oracle, we have that $x^{\mathrm{ch}}_{i,j}$ 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-PCPC-CMA game, then we have $Pr[j_i = j_i^{\mathrm{ch}}]$, for $i = 1, \ldots, m$. $\square$

6 Conclusion

In this paper, we presented the concept of proof-carrying proxy certificates. The idea is to generate the proxy certificate using a special signature scheme for which the validity of the generated signature proves the compliance of the signer with a credential-based policy. The proof adheres to the privacy principle of data minimization i.e. in the case where there exists multiple qualified sets of credentials for a policy, the proof does not reveal which specific set has been used to generate the signature. To implement our approach, we developed a concrete proof-carrying proxy certification scheme using bilinear pairings over elliptic curves. We defined formal security models for proof-carrying proxy certification schemes and proved the security of our construction under the defined models in the random oracle model. We are currently developing an experimental implementation framework for proof-carrying proxy certificates in the context of grid computing. The integration of well established credential standards (e.g. SPKI, SAML) is one of our goals. We are also working on improving the performance of our construction in terms of both computational and bandwidth consumption costs, and preparing and in-depth analysis of such costs. As discussed in the related work, an interesting line for future would be the construction of a proof-carrying proxy certification scheme based on the well known zero-knowledge proof of knowledge protocols.

References

1. A. Appel and E. Felten. Proof-carrying authentication. In *ACM Conference on Computer and Communications Security*, pages 52–62, 1999.
2. M. Backes, J. Camenisch, and D. Sommer. Anonymous yet accountable access control. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 40–46, New York, NY, USA, 2005. ACM Press.
3. 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.
4. 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.

5. J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett. Negotiating trust on the grid. In *2nd WWW Workshop on Semantics in P2P and Grid Computing*, New York, USA, May 2004.
6. J. Choi, K. Sakurai, and J. Park. Proxy certificates-based digital fingerprinting scheme for mobile communication. In *IEEE 37th Annual 2003 International Carnahan Conference on Security*, pages 587 – 594. IEEE Computer Society, 2003.
7. J. Claessens, B. Preneel, and J. Vandewalle. (how) can mobile agents do secure electronic transactions on untrusted hosts? a survey of the security issues and the current solutions. *ACM Trans. Inter. Tech.*, 3(1):28–48, 2003.
8. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — Crypto '86*, pages 186–194, New York, 1987. Springer-Verlag.
9. Organization for Economic Cooperation and Development (OECD). Recommendation of the council concerning guidelines governing the protection of privacy and transborder flows of personal data, 1980. http://www.oecd.org/home/.
10. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
11. 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.
12. B. Lee and K. Kim. Self-certified signatures. In *INDOCRYPT '02: Proceedings of the Third International Conference on Cryptology*, pages 199–214, London, UK, 2002. Springer-Verlag.
13. 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/.
14. G. Necula. Proof-carrying code. In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 106–119, New York, NY, USA, 1997. ACM Press.
15. B. Clifford Neuman. Proxy-based authorization and accounting for distributed systems. In *International Conference on Distributed Computing Systems*, pages 283–291, 1993.
16. 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.
17. N. Smart. Access control using pairing based cryptography. In *Proceedings CT-RSA 2003*, pages 111–121. Springer-Verlag LNCS 2612, April 2003.
18. S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. *RFC 3820*, June 2004.
19. F. Zhang and K. Kim. ID-based blind signature and ring signature from pairings. In *ASIACRYPT*, pages 533–547. Springer-Verlag LNCS 2501, 2002.
20. F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In *Public Key Cryptography*, pages 277–290, 2004.

A Proof of Theorem 1

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 $l_{i^\bullet, j^\bullet, k} \in \{1, \ldots, q_0\}$, for $k = 1, \ldots, m^\bullet_{i^\bullet, j^\bullet}$
3. Pick at random $\theta^\bullet_{i^\bullet, j^\bullet, k} \in \mathbb{Z}_q^*$, for $k = 2, \ldots, m^\bullet_{i^\bullet, j^\bullet}$, 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 $I^\bullet = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, n, H_0^\bullet, H_1^\bullet, H_2)$ be the global information, where the oracles $H_0^\bullet$ and $H_1^\bullet$ are controlled by algorithm $\mathcal{A}^\bullet$, H_2 is a public hash function, 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 the set of credential issuers $I = \{I_1, \ldots, I_N\}$ as follows: for $\kappa \in \{\kappa^\bullet_{i^\bullet, j^\bullet, k}\}$, the public key of I_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 I_κ is $R_\kappa = s_\kappa \cdot P$ for some randomly chosen $s_\kappa \in \mathbb{Z}_q^*$, (3) Run algorithm *User-Setup* to obtain a public/private key pair $(pk_\mathrm{f}, sk_\mathrm{f})$, (4) Give the global information $I^\bullet$ and the credential issuers' public keys $R_\kappa\}_{\kappa=1}^{N}$ to adversary $\mathcal{A}^\circ$.

Note. For $\kappa \in \{\kappa^\bullet_{i^\bullet, j^\bullet, k}\}$, the master key of I_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 $[A_\imath, H_{0,\imath}, \lambda_\imath]$ 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 $[A_\imath, H_{0,\imath}, \lambda_\imath]$, then return $H_{0,\imath}$
2. If $pk_u = pk_\mathrm{f}$ and A^{pk_u} does not appear on H_0^{list} and A^{pk_u} 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 $[A^{pk_u}, H_{0, l^\bullet_{i^\bullet, j^\bullet, 1}}, \mathsf{null}]$ to H_0^{list}
3. If $pk_u = pk_\mathrm{f}$ and A^{pk_u} does not appear on H_0^{list} and A^{pk_u} 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 $[A^{pk_u}, H_{0, l^\bullet_{i^\bullet, j^\bullet, k}}, r^{-1}_{\kappa^\bullet_{i^\bullet, j^\bullet, k}} \theta^\bullet_{i^\bullet, j^\bullet, k}]$ to H_0^{list}
4. Otherwise, pick at random $\lambda \in \mathbb{Z}_q^*$, return $\lambda \cdot P$ and add $[A^{pk_u}, \lambda \cdot P, \lambda]$ to H_0^{list}

Note. The random oracle $H_0^\bullet$ is such that $\tau^\bullet_{i^\bullet, j^\bullet} = \prod_{k=1}^{m^\bullet_{i^\bullet, j^\bullet}} e(R_{\kappa_{i^\bullet, j^\bullet, k}}, H_0(A^{pk_\mathrm{f}}_{i^\bullet, j^\bullet, k})) = e(P, ab \cdot P)$.

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 $\mathsf{Verify}(M_\mathrm{f}, \sigma_\mathrm{f}, pk_\mathrm{f}, Pol_\mathrm{f}) = \top$.

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^{pk_u}.

- ***CredGen-O***. Assume that adversary $\mathcal{A}^\circ$ makes a query on a tuple (I_κ, A). Let $[A_\imath, H_{0,\imath}, \lambda_\imath]$ be the tuple from H_0^{list} such that $A_\imath = A^{pk_u}$, then algorithm $\mathcal{A}^\bullet$ responds as follows:
  1. If $\imath = l^\bullet_{i^\bullet, j^\bullet, 1}$ and $\kappa \in \{\kappa^\bullet_{i^\bullet, j^\bullet, k}\}$, then report failure and terminate (event $\mathcal{E}_\mathsf{cred}$)
  2. If $\imath \neq l^\bullet_{i^\bullet, j^\bullet, 1}$ and $\kappa \in \{\kappa^\bullet_{i^\bullet, j^\bullet, k}\}$, then return $(r_\kappa \lambda_\imath) \cdot P_1 = (r_\kappa a) \cdot H_{0,\imath} = s_\kappa \cdot H_{0,\imath}$
  3. If $\kappa \in \{1, \ldots, N\} \setminus \{\kappa^\bullet_{i^\bullet, j^\bullet, k}\}$, then return $s_\kappa \cdot H_{0,\imath}$
- ***Sign-O***. Assume that adversary $\mathcal{A}^\circ$ makes a query on a tuple (M, Pol^{pk_f}). Algorithm $\mathcal{A}^\bullet$ responds as follows:

1. Pick at random $h_{i,1} \in \mathbb{Z}_q^*$, and $Y_{i,j} \in \mathbb{G}_1$ ($j = 1,\ldots,m_i$ and $i = 1,\ldots,m$)
2. Compute $\tau_{i,j} = \prod_{k=1}^{m_{i,j}} e(R_{\kappa_{i,j,k}}, H_0^\bullet(A_{i,j,k}^{pk_{\mathrm{f}}}))$ ($j = 1,\ldots,m_i$ and $i = 1,\ldots,m$)
3. Compute $x_{i,j+1} = e(P,Y_{i,j}) * \tau_{i,j}^{h_{i,j}}$, then compute $h_{i,j+1} = H_1(M\|x_{i,j+1}\|m\|i\|j+1)$. 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_1^{list}. If $(M,x_{i,j},m,i,j)$ appears on H_1^{list} in a tuple $[(M_\iota,x_\iota,m_\iota,i_\iota,j_\iota),H_{1,\iota}]$, then algorithm $\mathcal{A}^\bullet$ sets $h_{i,j} = H_{1,\iota}$. 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_1^{list}.
4. Let $x_{i,1} = e(P,Y_{i,m_i}) * \tau_{i,m_i}^{h_{i,m_i}}$ and $h_{i,1} = H_1(M\|x_{i,1}\|m\|i\|1)$, then
   (a) If $(M,x_{i,1},m,i,1)$ already appears on the list H_1^{list} in a tuple $[(M_\iota,x_\iota,m_\iota,i_\iota,1),H_{1,\iota}]$ such that $H_{1,\iota} \neq h_{i,1}$, 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_1^{list}.
5. Compute $Y = \sum_{i=1}^{m}\sum_{j=1}^{m_i} Y_{i,j}$ and $Z = (sk_u + H_2(Y))^{-1}\cdot P$ then return $([x_{i,j}]_{j=1}^{m_i}]_{i=1}^{m}, Y, Z)$ to adversary $\mathcal{A}^\circ$.

Algorithm $\mathcal{A}^\bullet$ controls the random oracle $H_1^\bullet$ as follows: assume that adversary $\mathcal{A}^\circ$ makes a query to the random oracle $H_1^\bullet$ 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_1^{list} in a tuple $[(M_\iota,x_\iota,m_\iota,i_\iota,j_\iota),H_{1,\iota}]$, then output $H_{1,\iota}$
2. Otherwise, pick at random $H \in \mathbb{Z}_q^*$, output H and add $[(M,x,m,i,j),H]$ to H_1^{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 $H_1^\bullet$. The success probability of adversary $\mathcal{A}^\circ$ in forging a valid ring signature scheme is then taken over the space $(w,H_1^\bullet)$. Let $\mathcal{S}$ be the set of successful executions of adversary $\mathcal{A}^\circ$, then the following holds

$$\mathrm{Adv}_{\mathcal{A}^\circ} = Pr[(w,H_1^\bullet)\in\mathcal{S}] \geq \varepsilon \tag{4}$$

Let $\mathcal{E}_0$ be the event that adversary $\mathcal{A}^\circ$ succeeds in forging the signature $\sigma_{\mathrm{f}} = ([x_{i,j}^{\mathrm{f}}]_{j=1}^{m_i}]_{i=1}^{m}, Y^{\mathrm{f}}, Z^{\mathrm{f}})$ without making a query to the random oracle $H_1^\bullet$ on at least one of the tuples $(M_{\mathrm{f}}, x_{i,j}^{\mathrm{f}}, m, i, j)$, and 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 *Sign-O*. Then, we have

$$Pr[\mathcal{E}_0] \leq \frac{m_{\vee\wedge}m_\vee}{q} \ \ , \ \ Pr[\mathcal{E}_0'] \leq \frac{m_{\vee\wedge}q_s q_1}{q} \tag{5}$$

Let $\mathcal{S}'$ be the set of successful executions of adversary $\mathcal{A}^\circ$ for which it has made queries to the random oracle $H_1^\bullet$ on the all the tuples $(M_{\mathrm{f}}, x_{i,j}^{\mathrm{f}}, m, i, j)$, then the following holds

$$Pr[(w,H_1^\bullet)\in\mathcal{S}'] = Pr[\neg\mathcal{E}_0].Pr[\neg\mathcal{E}_0'].Pr[(w,H_1^\bullet)\in\mathcal{S}] \geq (1-\frac{m_{\vee\wedge}m_\vee}{q}).(1-\frac{m_{\vee\wedge}q_s q_1}{q}).\varepsilon \tag{6}$$

Let $Q_1,\ldots,Q_{q_1}$ denote the different queries made by adversary $\mathcal{A}^\circ$ to the random oracle $H_1^\bullet$. We denote by $Q_{\beta_{i,j}}$ (for $\beta_{i,j}\in\{1,\ldots,q_1\}$) the query made by adversary $\mathcal{A}^\circ$ to the random oracle $H_1^\bullet$ on the tuple $(M_f, x^f_{i,j}, m, i, j)$. 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_1, this gives us a partition of $\mathcal{S}'$ in at least $q_1 + 1 - \beta$ classes.

Let $\mathcal{E}_1$ be the event that algorithm $\mathcal{A}^\bullet$ obtains a successful execution $(w^1, H_1^{\bullet 1}) \in \mathcal{S}'_{\beta^1_{i^{lq},j^{lq}}}$, for some last-query index $\beta^1_{i^{lq},j^{lq}}$, after invoking t_1 times adversary $\mathcal{A}^\circ$ with randomly chosen tuples $(w, H_1^\bullet)$. In the particular case where $t_1 = (Pr[w, H_1^\bullet) \in \mathcal{S}'])^{-1}$, and since $(1 - \frac{1}{X})^X \leq e^{-1}$ (for $X > 1$), the following statement holds

$$Pr[\mathcal{E}_1] = 1 - (1 - Pr[(w, H_1^\bullet) \in \mathcal{S}'])^{t_1} \geq 1 - e^{-1} > \frac{3}{5} \tag{7}$$

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

$$\mathcal{J} = \{\beta_{i^{lq},j^{lq}} \ s.t. \ Pr[(w, H_1^\bullet) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}} | (w, H_1^\bullet) \in \mathcal{S}'] \geq \gamma\}, \text{ where } \gamma = \frac{1}{2(q_1 + 1 - \beta)}$$

Let $\mathcal{S}'_{\mathcal{J}} = \{(w, H_1^\bullet) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}} \ s.t. \ \beta_{i^{lq},j^{lq}} \in \mathcal{J}\}$ 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

$$\begin{aligned} Pr[(w, H_1^\bullet) \in \mathcal{S}'_{\mathcal{J}} | (w, H_1^\bullet) \in \mathcal{S}'] &= \sum_{\beta_{i^{lq},j^{lq}} \in \mathcal{J}} Pr[(w, H_1^\bullet) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}} | (w, H_1^\bullet) \in \mathcal{S}'] \\ &= 1 - \sum_{\beta_{i^{lq},j^{lq}} \notin \mathcal{J}} Pr[(w, H_1^\bullet) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}} | (w, H_1^\bullet) \in \mathcal{S}'] \\ &\geq 1 - (\frac{1}{2\gamma} - |\mathcal{J}|).\gamma \geq \frac{1}{2} \end{aligned} \tag{8}$$

Let $\alpha = (1 - \frac{m_{\vee\wedge} m_{\vee}}{q}).(1 - \frac{m_{\vee\wedge} q_s q_1}{q}).\varepsilon.\gamma$, then equation (6) leads to the following statement

$$Pr[(w, H_1^\bullet) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}}] = Pr[(w, H_1^\bullet) \in \mathcal{S}'].Pr[(w, H_1^\bullet) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}} | (w, H_1^\bullet) \in \mathcal{S}'] \geq \alpha \tag{9}$$

The oracle $H_1^\bullet$ can be written as a pair $(\tilde{H}_1, h_{i^{lq},j^{lq}})$, where $\tilde{H}_1$ corresponds to the answers for all the queries to oracle $H_1^\bullet$ 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:

$$\Omega_{\beta_{i^{lq},j^{lq}}} = \{(w, (\tilde{H}_1, h_{i^{lq},j^{lq}})) \in \mathcal{S}' \ s.t. \ Pr_{h_{i^{lq},j^{lq}}}[(w, (\tilde{H}_1, h_{i^{lq},j^{lq}})) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}}] \geq \delta - \alpha\}$$

For $\delta = 2\alpha$ and according to the splitting lemma defined in [16], the following statements hold

$$\forall \ (w, (\tilde{H}_1, h_{i^{lq},j^{lq}})) \in \Omega_{\beta_{i^{lq},j^{lq}}}, \ Pr_{h_{i^{lq},j^{lq}}}[(w, (\tilde{H}_1, h_{i^{lq},j^{lq}})) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}}] \geq \alpha \tag{10}$$

$$Pr[(w, (\tilde{H}_1, h_{i^{lq},j^{lq}})) \in \Omega_{\beta_{i^{lq},j^{lq}}} | (w, (\tilde{H}_1, h_{i^{lq},j^{lq}})) \in \mathcal{S}'_{\beta_{i^{lq},j^{lq}}}] \geq \frac{\alpha}{\delta} = \frac{1}{2} \tag{11}$$

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

$$Pr[\mathcal{E}_2] = 1-(1-(\alpha-\frac{1}{q}))^{t_2} \geq 1-e^{-1} > \frac{3}{5} \tag{12}$$

Consider $(w^1, (\tilde{H}_1^1, h^1_{i^{lq},j^{lq}}))$ and $(w^1, (\tilde{H}_1^1, h^2_{i^{lq},j^{lq}}))$, 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 $H_1^\bullet$ to the queries of adversary $\mathcal{A}^\circ$ are identical only until the query $Q_{\beta^1_{i^{lq},j^{lq}}}$.

Let $\sigma_f^1 = ([[x^1_{i,j}]_{j=1}^{m_i^1}]_{i=1}^{m^1}, Y^1, Z^1)$ and $\sigma_f^2 = ([[x^2_{i,j}]_{j=1}^{m_i^2}]_{i=1}^{m^2}, Y^2, Z^2)$ 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 $m_i^1 = m_i^2 = m_i$ $(i = 1, \ldots, m)$. In this case, the following statements hold

1. $x^1_{i,j} = x^2_{i,j}$ $(j = 1, \ldots, m_i$ and $i = 1, \ldots, m)$
2. $h^1_{i,j} = h^2_{i,j}$ (for$j \neq j^{lq}$ and $i \neq i^{lq}$) and $h^1_{i^{lq},j^{lq}} \neq h^2_{i^{lq},j^{lq}}$

The fact that σ_f^1 and σ_f^2 are valid ring signatures leads to the equality $e(P, Y^2 - Y^1) = \tau_{i^{lq},j^{lq}}^{h^1_{i^{lq},j^{lq}} - h^2_{i^{lq},j^{lq}}}$. With probability greater than $1/q_0^{m_{\vee\wedge}m_\vee}$, we have $\tau_{i^\bullet,j^\bullet} = \tau_{i^{lq},j^{lq}}$. In this case, note that adversary $\mathcal{A}^\circ$ does not make a query to oracle *CredGen* on assertion $A_{l_{i^\bullet,j^\bullet},1}$ (event $\mathcal{E}_{\text{cred}}$ does not occur). This case leads to the equality $\tau_{i^{lq},j^{lq}} = e(P, ab \cdot P)$, and so $Y^2 - Y^1 = (h^1_{i^{lq},j^{lq}} - h^2_{i^{lq},j^{lq}}) \cdot (ab \cdot P)$. Thus, with probability $Pr[\mathcal{A}^\bullet$ wins$]$, algorithm $\mathcal{A}^\bullet$ succeeds to obtain $ab \cdot P$ by computing the quantity $(h^1_{i^{lq},j^{lq}} - h^2_{i^{lq},j^{lq}})^{-1} \cdot (Y^2 - Y^1)$. From statements (7), (8), (11) and (12), we have $\text{Adv}_{\mathcal{A}^\bullet} = Pr[\mathcal{A}^\bullet \text{ wins}] \geq \frac{9}{100 q_0^{m_{\vee\wedge}m_\vee}} \cdot \frac{1}{\sum_{l=1}^{m_\vee} l!\binom{m_\vee}{l}}$.

For $q \geq Max\{2m_{\vee\wedge}m_\vee, 2m_{\vee\wedge}q_s q_1\}$ and $\varepsilon \leq 32(q_1 + 1 - m_{\vee\wedge}m_\vee)/q$, the running time of adversary $\mathcal{A}^\bullet$ is such that the following holds

$$t_{\mathcal{A}^\bullet} = (t_1 + t_2).t_{\mathcal{A}^\circ} = (\frac{\gamma}{\alpha} + \frac{1}{\alpha - 1/q}).t_{\mathcal{A}^\circ} \leq (\frac{4}{\varepsilon} + \frac{32(q_1 + 1 - m_{\vee\wedge}m_\vee)}{\varepsilon}).t_{\mathcal{A}^\circ} \leq \frac{32q_1 + 4}{\varepsilon}.t_{\mathcal{A}^\circ}$$