Attribute Based Signatures: The Case for Automation

Lalitha Muthu Subramanian and Roopa Vishwanathan
Department of Computer Science, New Mexico State University, Las Cruces, New Mexico, U.S.A.

Keywords: Attribute-based Signatures, Groth-Sahai Proofs, Computer-aided Cryptography.

Abstract: Attribute-based signatures (ABS) allow a signer to sign boolean predicates using certain attributes that are accepted by signing policies and are associated with signing keys. Ideally, the verifier of the signature must not obtain any other information except that a signer possessing the right attributes produced the signature. The goal of an attribute-based signature is to preserve the anonymity of the signer and the signers' attributes, while ensuring that a signature will only pass verification if the signer possesses enough valid attributes to satisfy the given predicate. In this paper, we explore the question of what would it take to automate the construction and proofs of complex cryptographic protocols such as attribute-based signatures. We posit that, at the minimum, it would require: (1) creating new data types to support attribute-based cryptography, and more generally, pairing-based cryptography (2) creating new function templates to operate on the data types (3) creating libraries for *credential bundles*, which are commonly used to tie in a set of attributes to a single user (4) automating Groth-Sahai witness-indistinguishable proofs, and (5) automated functional support for constructing a general framework for attribute-based signatures.

1 INTRODUCTION

Attribute based Signatures (Escala et al., 2011; El Kaafarani et al., 2014; Maji et al., 2011) have attracted significant interest primarily due to the fine-grained privacy control options they offer when compared to other well-used techniques, such as traditional digital signatures, ring and mesh signatures. Attribute Based signatures (ABS) allows a signer owning a set of attributes to anonymously sign a message, without revealing the attributes that were used to generate a valid signature. Thus, in ABS, the recipient of a signature is convinced that someone with a set of right attributes has signed a predicate representing a signing policy, and indeed authenticated a given message, but the recipient neither learns the signer's identity nor the attributes used to satisfy the predicate. Hence, attribute based signatures offer the signer a privacy-preserving environment where valid signatures can only be produced by users who hold a subset of valid attributes that satisfy some signing policy.

2 RELATED WORK

Halevi (Halevi, 2005), first proposed the idea of creating automated tools for cryptographic *proof* verification. His vision was to automate the tools that help with commonly used argument techniques, i.e., tools that help with canonical, standard parts of the proof, and leave the more subtle, idiosyncratic parts of the proof to the prover to be proved manually. Halevi's paper stressed the need for such tools to be designed to work with existing proof frameworks, i.e., frameworks that cryptographers understand and are familiar with, as opposed to proposing new proof frameworks, and to work in the widely-used computational model of cryptography, as opposed to the symbolic model of Dolev and Yao (Dolev and Yao, 1983). As motivational examples, Halevi presented two case-studies, in which two manually-written proofs, one for a new block cipher encryption mode, and the other for the Cramer-Shoup encryption system, are analyzed and pointed out places where a hypothetical automated tool could possibly have been very useful in the process of constructing the proof. In response to this, recently researchers have proposed automated tools such as EasyCrypt (Barthe et al., 2011; Barthe et al., 2013b), CryptoVerif (Blanchet, 2006), CertiCrypt (Barthe et al., 2009), ZooCrypt (Barthe et al., 2013a), and the more general-purpose proof assistant Coq (labs,) augmented with the Foundational

Subramanian, L. and Vishwanathan, R.
Attribute Based Signatures: The Case for Automation.
DOI: 10.5220/0006901405370542
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 537-542
ISBN: 978-989-758-319-3

Cryptography Framework library (FCF) (Petcher and Morrisett, 2015). Most of these tools are proposed for *proof automation*, not *protocol construction*. Certicrypt, EasyCrypt and FCF can be used to verify some primitives as well as most protocols, but the proofs are non-intuitive and tedious to construct. ZooCrypt could be used to verify proofs of cryptographic primitives, but works only for a special class of constructs: padding-based encryption schemes. In the area of pairing-based proof automation, there has been work in automating proofs in the generic group model (Barthe et al., 2014; Ambrona et al., 2016; Ambrona et al., 2017), and the standard model (Barthe et al., 2015).

Most of these tools have been proposed for generating proofs for simple cryptographic primitives, such as ElGamal (Barthe et al., 2011), padding-based encryption schemes, etc. Automating constructions of advanced cryptographic protocols such as attribute-based signatures, or attribute-based encryption is a non-intuitive and challenging task. But it is nevertheless useful, since many of the fundamental mathematical constructs, and cryptographic operations used in the design of ABS schemes are also used by other important categories of cryptographic protocols, such as zero-knowledge proofs, identity-based encryption, and more. Hence, work in this area will be impactful beyond just ABS schemes.

2.1 Contribution

In this position paper, we compare techniques used to construct attribute-based signature schemes across the literature, and identify those building blocks, assumptions, and primitives that are commonly used and would be suitable candidates for automation. Also, we explore possible ways to achieve this.

3 AUTOMATION SUPPORT

In this section, we give the common cryptographic primitives, assumptions, and data types that are used in the construction of ABS schemes, and propose templates for their automation.

3.1 Common Primitives

We now enlist some common functional primitives that are used in the construction of ABS schemes; any tool that automates the construction of ABS schemes would need to provide templates and libraries that implement these primitives.

A new generic primitive called *credential bundles* (Maji et al., 2011) is widely used in ABS constructions. Credential bundles, intuitively tie in a user's identity to a set of attributes, and prevent collusion among users potentially possessing the same kind of attribute (e.g., employee), and hence model the requirements of a set of publicly verifiable attributes issued to a unique signer that resist collusion. A credential bundle scheme has a message space m, and consists of the following algorithms.

- CB.Setup: On input a security parameter, outputs a verification key V_k and a secret key S_k.
- CB.Gen: On input $(S_k, \{m_1, m_2, ..., m_n\} \subseteq M)$ (where $\{m_1, \ldots, m_n\}$ is a set of messages belonging to message-space M), outputs a tag τ and signatures $\sigma_1, ..., \sigma_n$.
- CB.Ver: On input $(V_k, m_i, (\tau, \sigma_i))$, outputs a boolean value.

This scheme is right if $\text{CB.Ver}(V_k, m_i, (\tau, \sigma_i)) = 1$ for all i.

A credential bundle scheme can easily be instantiated with any digital signature scheme where there is a collection of signatures, of messages $\tau||m_i$, where each m_i is a bitstring representing an attribute and τ is an identifier that is unique to each user. Also, in the above scheme, one could remove some of the σ_i's from an existing credential bundle to create a new bundle on a subset of messages/attributes. Having an exclusive library for credential bundles in the automation of an attribute-based signature scheme would prove extremely useful since little effort would be involved in creating attributes and bundles, such as those required to align with the semantics of the ABS scheme. Maji et al., (Maji et al., 2011) have used the Boneh Boyen (Boneh and Boyen, 2004), and Waters (Waters, 2005) signature schemes as the credential bundle component in their ABS scheme. The attributes in their credential bundle scheme are simple Boneh Boyen/Waters signatures on messages of the form "userid || attr". Although only Boneh Boyen, and Waters' signature schemes have been used in (Maji et al., 2011), we can essentially have the general framework of ABS to support any other signature schemes as the credential bundles, such as Boneh-Boyen-Shacham signatures, and more (Boneh et al., 2004; Boyen and Waters, 2006). Furthermore, creating a general template for credential bundles is useful since, although existing ABS scheme use a bundle which ties its attributes to a user identity, one could potentially imagine a bundle that ties in a *set of functions* to a user identity. This generalized version of a credential bundle would be very useful in automating constructions of functional en-

cryption (Datta et al., 2018).

Another primitive fundamentally used in the construction of ABS schemes is non-interactive witness indistinguishability (NIWI) proofs. NIWI proofs are used to commit to a set of attributes that purportedly satisfy a boolean predicate while creating an attribute-based signature. The NIWI proof system is comprised of algorithm mentioned below (Groth and Sahai, 2008).

- NIWI.Setup: On input a security parameter, this outputs a reference string crs.
- NIWI.Prove: Given an input (crs, ϕ, x), where ϕ is a boolean formula and $\phi(x) = 1$, output a proof π.
- NIWI.Verify: On an input of (crs, ϕ, π), either accept or reject the proof.

3.2 Data Type/Function Support

Automating the construction of attribute based signature schemes involves creating abstract data types for pseudo attributes and pseudo predicates, both of which are used in the attribute generation and signing phases.

- A type for a pseudo attribute A' denoted as P_A.
- A type for pseudo predicate, $\tilde{\Upsilon}$ denoted as P_P.

The functions to support the new data types are:

- $(\tau, \sigma_a; a \in A) \leftarrow \mathsf{Parse}(SK_A)$: This function parses a secret key, SK_A as $(\tau, \sigma_a, (a \in A))$, where a is an attribute belonging to an attribute set A.
- $\tilde{\Upsilon} \leftarrow \mathsf{PPCreate}(a \in A, \Upsilon)$: This function takes in an attribute a, and a boolean predicate Υ, and creates a pseudo predicate $\tilde{\Upsilon} \in P_P$.
- $(a_m\Upsilon) \leftarrow \mathsf{PACreate}(a \in A, \Upsilon)$: This function takes in an attribute a, a boolean predicate Υ, and returns a pseudo-attribute $a_m\Upsilon \in P_A$.

We envision the above functions as abstract data types since the key feature of such data types is that they are characterized by the operation that is performed on them, and do not specify any concrete implementation, thus giving us the flexibility to instantiate them with any implementation of our choice. Abstract data types are accessed through methods and support multiple implementations too. In general the widely used methods for an abstract data type may include initialization, addition or removal of data, and of retrieval data.

Array, List, Map, Queue, Set, Stack, Table are some of the most commonly used abstract data types in high-level languages. A stack for example allows the functions empty(), isEmpty(), push(), top(), pop() to be performed (without specifying the implementation) when defined as an abstract data type: empty:: Stack a, isEmpty:: Stack a $\longrightarrow$ Bool, push:: a $\longrightarrow$ Stack a $\longrightarrow$ Stack a, top :: Stack a $\longrightarrow$ a, pop:: Stack a $\longrightarrow$ (a,Stack a)

In a similar vein, defining PPCreate and PACreate as abstract data types allows us to represent attributes and predicates as data structures, without having to tie down an automated tool's libraries to a specific implementation. Possible implementations for a PPCreate data type include a 2-d array/linked list, or a matrix. Possible implementations for a PACreate data type include a singly/double linked list, or a 1-d array.

3.3 Monotone Span Programs

Consider a Monotone boolean function $\Upsilon : \{0,1\}^n \longrightarrow 0,1$. The Monotone Span Program for Υ over a field F is defined as a matrix $\mathbf{M}$ containing the entries in F, alongside a labeling function defined as $a : [l] \longrightarrow [n]$ such that it relates each row of $\mathbf{M}$ with an input variable of Υ, for every $(x_1, ..., x_n) \in \{0,1\}^n$.

$$\Upsilon(x_1, ..., x_n) = 1 \Longleftrightarrow \exists \vec{v} \in F^{1 \times l} : \vec{v}\mathbf{M} = [1, 0, ..., 0] \quad (1)$$

and $\big(\forall i : x_{a(i)} = 0 \Rightarrow v_i = 0\big)$.

With $\mathbf{M}$ being the monotone matrix of length l and width of the span program as t, we propose to create a data type called **MSP**, that represents a monotone span program, which can be declared as follows:

$$\mathbf{MSP}\ \mathbf{M}[l \times t] = \text{new}\ \mathbf{M}[l][t]$$

where l and t denote the rows and columns of the **MSP** matrix. We propose to create the following functions for monotone span program matrix:

- $\tilde{\mathbf{M}} \leftarrow \mathsf{set}(\mathbf{M}, i, j, a_{ij})$: This function is used to set a value $a_{i,j}$ in the matrix of type **MSP**, where i and j denote the row and column location respectively. This function outputs $\tilde{\mathbf{M}}$ with the updated values.
- $\{0,1\} \leftarrow \mathsf{checkVector}(\mathbf{M}, v)$: This function is used to check if a vector, $v = [1, \ldots, 0]$ is present in the any of the l rows of the matrix $\mathbf{M}$.
- $a_{i,j} \leftarrow \mathsf{get}(\mathbf{M}, i, j)$: This function is used to get a value $a_{i,j}$ in the matrix of type **MSP**, where i and j denotes the element in the matrix.
- $\{0,1\} \leftarrow \mathsf{span}(\mathbf{M}, v)$: This function is used to check if a vector v spans any of the columns of the matrix $\mathbf{M}$.

4 GENERAL FRAMEWORK

Let A denote the universe containing the ABS attributes. Let $A \cap A'$ denote the pseudo attributes sets where, $A \cap A' = \emptyset$. For each message represented by m, and a claim predicate Υ, the pseudo attributes that are associated with (m, Υ) are represented by $a_m \Upsilon A'$. Let *CB* denote a credential bundle scheme.

- $\Phi[V_k, m, \Upsilon]$: This is the pairing expression which represents part of the input of the non-interactive witness indistinguishability proof, NIWI.Prove, that goes into the construction of an attribute-based signature, and typically needs to be encoded into the Groth-Sahai proof system.
- $Com(\cdots)$: This represents the commitment function that is defined in the Groth-Sahai proof system to create a commitment to the values of witnesses.
- $S^v C^\tau, D^\tau$: These are intermediate values to which the prover commits to in the Groth-Sahai proofs. $S^v \in \mathbb{H}$, $C^\tau, D^\tau \in \mathbb{G}$, where $\mathbb{G}, \mathbb{H}$ are groups.

The generic construction of ABS with a single attribute-issuing authority parameterized with a universe of possible attributes A and message space M, consists of the following five algorithms as defined by Maji *et al.* (Maji et al., 2011):

- **ABS.TSetup**: This step is to generate the public reference information TPK.
- **ABS.Setup**: This algorithm generates a key pair APK, ASK.
- **ABS.AttrGen**: This outputs a signing key SK_A corresponding to a set of attributes A.
- **ABS.Sign**: On input $(PK = (TPK, APK), SK_A, m \in M, \Upsilon)$, where $\Upsilon(A) = 1$, outputs a signature σ.
- **ABS.Ver**: On input $(PK = (TPK, APK), m, \Upsilon, \sigma)$, outputs a boolean value.

We propose in this paper, to create automated libraries that would enable the signer and verifier to perform the above mentioned functionalities sequentially. This entails creating a template library that is agnostic of the specifics of the public-key encryption scheme used for the **ABS.TSetup**, and **ABS.ASetup** algorithms (can generate key-pairs for *any* scheme), creating a credential bundle library that is independent of the signature scheme used (either Boneh-Boyen (Boneh and Boyen, 2004), Boneh-Boyen-Shacham (Boneh et al., 2004), etc.), and more ambitiously, creating a library defining a non-interactive proof system, that can create commitments to attributes used in ABS, using *any* non-interactive witness indistinguishable proof system, e.g., Groth-Sahai (Groth and Sahai, 2008).

4.1 Groth-Sahai Proof System

The Groth-Sahai proof system (Groth and Sahai, 2008) is the primary method for instantiating the non-interactive witness indistinguishable argument in an ABS scheme. While the Groth-Sahai method can be used to prove witnesses to any general statement, we are interested only in non-interactive proofs, where the witness is an attribute that satisfies a Boolean predicate. These proofs work by giving a commitment to the witness values and then proving that these committed values satisfy the pairing equations. For generating Groth-Sahai commitments in an ABS scheme, the statement $\Phi[V_k, m, \Upsilon]$ needs to be efficiently encoded. Let us assume we need to commit a group $\mathbb{Z}$ (which could be either $\mathbb{G}$ or $\mathbb{H}$, and will be clear from the context), $\mathbb{Z}$ becomes the formal variable representing the commitment. In this section, we briefly go over the Groth-Sahai proof system and its application to ABS (Maji et al., 2011).

Let us consider the pseudo-predicate $\tilde{\Upsilon}$ to be a matrix of data type **MSP** defined in section 3.3. Let the size of $\tilde{\Upsilon}$ be $l \times t$,with the ith row corresponding to the a(i)-th attribute. In order to establish the statement $\Phi[V_k, m, \Upsilon]$,the following two equations should be proved:

$$\exists \tau, \sigma_1, \ldots, \sigma_n, v_1, \ldots\ldots, v_n : \vec{v}\mathbf{M} = [1, 0, \ldots, 0] \quad (2)$$

$$\bigwedge_{i=1}^{l} \left[v_i \neq 0 \Rightarrow CB.Ver\left(vk, a_{a(i)}, \left(\tau, \sigma_{a(i)}\right)\right) = 1 \right] \quad (3)$$

In addition to the above equation, the signer would have to commit to vector $\vec{v}$ which can be computed from his assignment of $\tilde{\Upsilon}$. Once the signer commits to vector $\vec{v}$, the new boolean expression formed is a combination of the two kinds of clauses. One kind of clause has the following form

$$\exists \vec{v} : \vec{v}\mathbf{M} = [1, \ldots., 0] \quad (4)$$

Committing to the values g^{v_i} and proving the following pair of equations for each $j \in [t]$ will prove that the boolean expression formed by committing to the values is a combination of the clauses.

$$\prod_{i=1}^{l} e\left(Com\left(g^{v_i}\right), h^{\mathbf{M}_{i,j}}\right) = \begin{cases} (g, h) & \text{if } j = 1 \\ \left(g^0, h\right) & \text{otherwise} \end{cases} \quad (5)$$

The second clause has the following form:

$$\exists \tau, \sigma, v : [v \neq 0 \Rightarrow CB.Ver(vk, m, (\tau, \sigma)) = 1] \quad (6)$$

The clause in Equation 6 can also be written as:

$$\exists \tau, \sigma, v : [v \neq 0 \Rightarrow DS.Ver(vk, \tau||m, \sigma) = 1] \quad (7)$$

If we have $(\tau, \sigma = (S, r), v)$ as a witness to the above expression, the Groth-Sahai system then expresses τ bitwise as $\tau = \Sigma_i \tau_i 2^i, \tau||m$ and r is expressed as $r = \Sigma_i r_i 2^i$.

We create a commitment to both r_i, τ_i in both groups as $g^{r_i}, h^{r_i}, g^{\tau_i}, h^{\tau_i}$, and then prove that each of them is a single bit by using the following pairing equations:

$$e(Com(g^{r_i}), h) = e(g, Com(h^{r_i})) \quad (8)$$

$$e(Com(g^{\tau_i}), h) = e((g, Com(h^{\tau_i})) \quad (9)$$

$$e(Com(g^{r_i}), Com(h^{r_i})) = e(Com(g^{r_i}), h) \quad (10)$$

$$e(Com(g^{\tau_i}), Com(h^{\tau_i})) = e(Com(g^{\tau_i}), h) \quad (11)$$

The prover commits to some of the intermediate values $S^v \in H$ and $C^\tau, D^r \in G$, since he cannot directly compute $BC^{\tau||m}D^r$ or S^v given the committed values; hence he proves the following equations:

$$e(Com(D^r), h) = \prod_i e\left(D^{2^i}, Com(h^{r_i})\right) \quad (12)$$

$$e(Com(g^v), Com(S)) = e(g, Com(S^v)) \quad (13)$$

$$e(Com(C^\tau), h) = \prod_i e\left(C^{2^i}, Com(h^{\tau_i})\right) \quad (14)$$

$$e(Com(g^v), h) = e\left(BC^{2^{|\tau|}m}, Com(S^v)\right) e(Com(C^\tau), Com(S^v))\, e(Com(D^\tau), Com(S^v)) \quad (15)$$

The coefficients in the above equations are computed publicly.

4.2 Automation of General Framework

Automating the general framework of ABS involves creation of separate libraries for data types and core functions. The next step would be to set up the credential bundles and use *some* signature scheme, e.g., the Boneh Boyen scheme for the construction of ABS. This automated library can be named as CB. We propose the following core functions as a part of automating the general framework of ABS Scheme.This is efficiently encoded in the Groth-Sahai proof system by using the 'commit' operation defined in section 4.1

- **ABS.TSetup**: This library is created such that it enables the signature trustee to run *crs* with **NIWI**.Setup and (tvk, tsk) and publishes $TPK = (crs, tvk)$ as the outcome
- **ABS.ASetup**: This library is created to perform the functions of the attribute issuing authority and publishes APK as avk and sets ASK as ask.
- **ABS.AttrGen**: This library essentially outputs the result of $\mathsf{CB.Gen}(ask, A)$.
- **ABS.Sign**: This library performs the signing operation by creating the boolean expression $\Phi[V_k, m, \Upsilon]$ and computing $\exists \tau, \sigma_1, \ldots, \sigma_n, v_1, \ldots, v_n$. This template plays a major part in computing π which is used in the creation of the signature using NIWI scheme. This is done by proving the the signature. NIWI.Prove$(crs, \Phi[V_k, m, \Upsilon]; (\tau, \sigma_1, \ldots, \sigma_n))$
- **ABS.Ver**: This library uses the NIWI proof to compute the output by verifying NIWI.Verify$(crs, \Phi[V_k, m, \Upsilon]; \pi)$. This is verified using the Groth-Sahai proof system as explained in section 4.2 from bilinear pairing equations and performing a commit operation.

The automation of Groth-Sahai proofs will involve the following method of creating templates. The major automation effort would be to perform the commitment operations in the Groth Sahai NIWI proofs. The envisioned tool is expected to make the operation successful by involving a very little manual effort in verifying the signature. It can be correlated to having a "commit" button, and the system performs the operation and verifies the signature once the "commit" button is pressed, returning 1 if true, 0 otherwise.

5 FUTURE WORK AND CONCLUSION

In this paper, we have outlined the components of a tool that we propose to build, that will assist in the automation of the construction of attribute-based signatures. Developing such a tool is a challenging task, but will be very useful, given that the construction of ABS schemes is a tedious, error-prone task. Furthermore, the fundamental mathematical assumptions and basic cryptographic constructs such as span programs, and non-interactive witness indistinguishable proofs that are used in ABS, are used across the spectrum of cryptographic protocols, well beyond the realm of ABS. Hence, any effort towards this goal is likely to benefit other areas within cryptography

too. As a starting point, we have outlined the basic data types, functions to manipulate those data types, and proposed a general library template that can construct ABS scheme automatically. Building upon this preliminary work will require not just implementing the ideas presented in this paper, but also borrowing knowledge and theories from the logic and automated reasoning communities to build a tool that not only produces correct constructions, but is also grounded in a rigorous theoretical framework.

REFERENCES

Ambrona, M., Barthe, G., Gay, R., and Wee, H. (2017). Attribute-based encryption in the generic group model: Automated proofs and new constructions. In *Proceedings of the 2017 ACM CCS*, pages 647–664.

Ambrona, M., Barthe, G., and Schmidt, B. (2016). Automated unbounded analysis of cryptographic constructions in the generic group model. In *Advances in Cryptology - EUROCRYPT 2016*, pages 822–851.

Barthe, G., Crespo, J. M., Grégoire, B., Kunz, C., Lakhnech, Y., Schmidt, B., and Zanella-Béguelin, S. (2013a). Fully automated analysis of padding-based encryption in the computational model. In *Proceedings of the 2013 ACM CCS*, CCS '13, pages 1247–1260.

Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., and Strub, P. (2013b). Easycrypt: A tutorial. In *Foundations of Security Analysis and Design FOSAD*, pages 146–166.

Barthe, G., Fagerholm, E., Fiore, D., Mitchell, J. C., Scedrov, A., and Schmidt, B. (2014). Automated analysis of cryptographic assumptions in generic group models. In *Advances in Cryptology - CRYPTO*, pages 95–112.

Barthe, G., Grégoire, B., Heraud, S., and Béguelin, S. Z. (2011). Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology - CRYPTO*, pages 71–90.

Barthe, G., Grégoire, B., and Schmidt, B. (2015). Automated proofs of pairing-based cryptography. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security, CCS*, pages 1156–1168.

Barthe, G., Grégoire, B., and Zanella-Béguelin, S. (2009). Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT POPL 2009*, pages 90–101. ACM.

Blanchet, B. (2006). A computationally sound mechanized prover for security protocols. In *2006 IEEE Symposium on Security and Privacy (S&P)*, pages 140–154.

Boneh, D. and Boyen, X. (2004). Short signatures without random oracles. In *Eurocrypt*, pages 56–73. Springer.

Boneh, D., Boyen, X., and Shacham, H. (2004). Short group signatures. In *Crypto*, pages 41–55. Springer.

Boyen, X. and Waters, B. (2006). Compact group signature without random oracles. In *Eurocrypt*, pages 427–444. Springer.

Datta, P., Okamoto, T., and Tomida, J. (2018). Full-hiding (unbounded) multi-inner product functional encryption from the *k*-linear assumption. In *To appear in PKC*.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207.

El Kaafarani, A., Ghadafi, E., and Khader, D. (2014). Decentralized traceable attribute-based signatures. In Benaloh, J., editor, *Topics in Cryptology – CT-RSA 2014*, pages 327–348, Cham. Springer International Publishing.

Escala, A., Herranz, J., and Morillo, P. (2011). Revocable attribute-based signatures with adaptive security in the standard model. In Nitaj, A. and Pointcheval, D., editors, *Progress in Cryptology – AFRICACRYPT 2011*, pages 224–241, Berlin, Heidelberg. Springer Berlin Heidelberg.

Groth, J. and Sahai, A. (2008). Efficient non-interactive proof systems for bilinear groups. In *Eurocrypt*, pages 415–432.

Halevi, S. (2005). A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181.

labs, I. The Coq proof assistant. https://coq.inria.fr/.

Maji, H. K., Prabhakaran, M., and Rosulek, M. (2011). Attribute-based signatures. In Kiayias, A., editor, *Topics in Cryptology – CT-RSA 2011*, pages 376–392, Berlin, Heidelberg. Springer Berlin Heidelberg.

Petcher, A. and Morrisett, G. (2015). The foundational cryptography framework. In *Principles of Security and Trust - 4th International Conference, POST, Proceedings*, pages 53–72.

Waters, B. (2005). Efficient identity-based encryption without random oracles. In *Eurocrypt*, pages 114–127. Springer.