Exploring Automation in Proofs of Attribute-based Encryption in the Standard Model[*]

Guruprasad Eswaraiah, Lalitha Muthu Subramanian, Roopa Vishwanathan
Department of Computer Science, New Mexico State University, USA
{guru,lalitha,roopav}@nmsu.edu

***Abstract*—Motivated by the complexity of cryptographic proofs, we propose methods to automate the construction and verification of cryptographic proofs in the *standard* model. Proofs in the standard model (as opposed to the random oracle model) are the gold standard of cryptographic proofs, and most cryptographic protocols strive to achieve them. The burgeoning complexity of cryptographic proofs implies that such proofs are prone to errors, and are hard to write, much less verify. In this paper, we propose techniques to generate automated proofs for attribute-based encryption schemes in the standard model, building upon a prototype tool, AutoG&P due to Barthe *et al.* In doing so, we significantly expand the scope of AutoG&P to support a rich set of data types such as multi-dimensional arrays, and constructs commonly used in cryptographic protocols such as monotone-access structures, and linear secret-sharing schemes. We also provide support for a extended class of pairing-based assumptions. We demonstrate the usefulness of our extensions by giving automated proofs of the Lewko *et al.* attribute-based encryption scheme, and the Waters' ciphertext-policy attribute-based encryption scheme.**

***Index Terms*—cryptography, attribute-based encryption, proof automation**

I. Introduction

Cryptographic schemes are designed and proven secure in rigorous frameworks, and the proof involves showing that a given scheme satisfies some notion of security, e.g., proving that encryption schemes are secure against chosen-plaintext (IND-CPA) or chosen-ciphertext (IND-CCA) attacks, proving that a signature scheme is existentially unforgeable against chosen-message attacks (EUF-CMA), proving a group signature scheme is anonymous and traceable, proving that a hash function is strongly/weakly collision-resistant, and more. Complex cryptographic protocols are designed from primitives such as encryption and signatures schemes, pseudorandom functions, etc. Writing proofs for primitives and protocols is a subtle, onerous, non-trivial process, and is known to be susceptible to errors.

Encryption, signature schemes, and a host of other cryptographic protocols that are built on them are constructed by hinging their security on one or more computational hardness assumptions, ranging from the simple discrete logarithm, decisional Diffie-Hellman (DDH), computational Diffie-Hellman (CDH) assumptions, to more involved ones such as q-strong Diffie-Hellman (qSDH), bilinear Diffie-Hellman (BDH), and decision linear (DLIN) assumptions, to name a few.

[*]Research supported by NSF award #1800088. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

The typical way in which a given scheme is proven secure is by reducing the hardness of breaking the scheme to one or more hardness assumptions, a technique which is usually termed a *reduction*. The reduction is usually intricate and detailed, and subtle errors could lead to the cryptographic scheme being rendered insecure. Several cryptographic protocols have been proven secure in the past, and the proofs have later been found to be inaccurate, with reasoning flaws, such as the errors in the proof of the well-used RSA-OAEP, pointed out by Shoup. To remedy this situation, it is imperative to invest efforts into the area of *automation* of cryptographic proofs. Fundamentally, any automated tool meant for the construction and verification of cryptographic proofs should be constructed so that it is possible for provers to prove statements in familiar, well-known models, such as IND-CPA/CCA, EUF-CMA, etc., without modifying or re-working the fundamental models, and the goal of the proof assistant should be to reduce the amount of time and effort it takes to manually develop a proof.

II. Related Work

Initial efforts focused on building tools for the *symbolic* model of cryptography [1], but almost all cryptographic protocols are proven secure in the *computational* model. Many recent tools that work in the computational model are EasyCrypt [2], CryptoVerif [3], and the more general-purpose proof assistant Coq [4] augmented with the Foundational Cryptography Framework library (FCF) [5]. In this paper, We focus on the practically relevant, *standard* model. Akinyele *et al.* [6] proposed new automated tools for pairing-based encryption and signature schemes. AutoG&P proposed by Barthe *et al.* [7] is a prototype tool that can analyze the security of pairing-based cryptographic primitives and implements a set of high-level logic rules for performing reductions but lacks support for several data structures, computational assumptions, and proof techniques.

Recent work by Eswaraiah *et al.* [8] shows how to automate proofs of *signature schemes* and eventually use the proofs of these signature schemes as building blocks in automating proofs of higher-level constructs such as attribute-based signature schemes (ABS), although they do not provide any details about this possible ABS extension. Their work fundamentally differs from ours in that they focus on *signature* schemes, while we focus on *encryption* schemes.

Attribute-based encryption originated as a form of Identity-based Encryption (IBE), first proposed by Sahai and Waters.

We focus on automating 1) Waters' scheme [9], which was the first to present a ciphertext-policy attribute-based encryption scheme in the standard model, under non-interactive assumptions, and 2) Lewko *et al.* [10] scheme, which was the first to propose a ciphertext-policy attribute-based encryption system, that is proven adaptively secure. Our choice of automating their proofs stems from the fact that both of these are considered landmark results in the ABE domain in the *standard model*, as well as that their proofs are considered challenging, and similar techniques can be used to automate proofs of other ABE schemes, with less involved proofs.

A. Contributions

Our main contribution is a formal logic to reason about the security of cryptographic primitives such as attribute-based encryption in the *standard* model, that are based on involved pairing-based assumptions, and use constructs that are not supported by existing proof assistants. Our logic is based on AutoG&P's existing rules and semantics, but extends the original tool to include rules for reasoning about games that can discharge proof obligations about security proofs that use structures such as monotone access structures and linear secret sharing schemes. Using this, we automate proofs of two efficient ciphertext-policy attribute-based encryption schemes: the Lewko *et al.* [10], and the Waters' scheme [9]. Both of these proofs are fairly involved, challenging, and are in the *standard* model. Using these as templates, one can potentially automate proofs of *any* other attribute-based encryption scheme.

III. Our Logic Rules

In order to build support for monotone access structures, access tress, and linear secret sharing schemes, we need to formalize logic rules for handling multi-dimensional arrays. We need to guide the tool on how to handle read/write operations in arrays, how to tell if two arrays are equal, and support simple relation-checks such as transitivity, symmetry of arrays. We need to guide the tool according to well-known logic rules, and help the tool draw inferences based on these rules. For example, checking if two arrays are equal, or checking if the span of a row or a subset of rows in a matrix representing a monotone span program is $[1, 0, 0, \ldots, 0]$. The latter is a fundamental operation used in construction of attribute-based encryption schemes.

In this section, we give our core logic rules for multidimensional arrays that are used to represent monotone access structures. Our rules are based on the theory of arrays by McCarthy, and theory combination methods proposed in the formal logic literature. In Figure 1, the extensionality rule, Ext, formalizes that if two n-dimensional arrays are equal in a security experiment, then the values they store at each index $i; i \in [1..n]$ will be the same. Note that every value from $[1..n]$ has to be at the same index in both arrays. A special case of this rule would be 2-dimensional arrays such as matrices and monotone access structures, where each matrix entry at all indices $(i, j); i, j \in [1..n]$ should be the same for two matrices a and b. The read over write rule, R-over-W, writes a value v to an index i in an n-dimensional array a, then reads the value at index i' in the same dimension. If $i = i'$, then it returns v, else it reads and returns the result of $\mathsf{read}(a, i')$. The Partial-Eq rule states that if two n-dimensional arrays a and b agree at every index, except those in the set $\mathcal{I}$, then for every $i \notin \mathcal{I}$, $\mathsf{read}(a, i)$ should be equal to $\mathsf{read}(b, i)$. The write with elimination rule, W-Elim eliminates write expressions, replacing them with reads. The Trans rule formalizes transitivity between three n-dimensional arrays a, b, c. The Subst rule replaces all occurrences of an n-dimensional array a with an n-dimensional array b in a game. The Symm rule states that if an n-dimensional array a stores the same value as n-dimensional array b on all indices, except those indices in set $\mathcal{I}$, then b also stores the same value on all indices as a, except those indices in set $\mathcal{I}$. We note that the Trans and Symm rules will be the same, regardless of the dimensionality of an array. We now state a soundness theorem for our rules. The proof of the following theorem is given in the full version of the paper.

Theorem 3.1: The conclusion of each rule in Definition 3.1 is satisfiable, if and only if one of its premises is satisfiable.

IV. General Structure of CP-ABE systems

The general structure of ciphertext policy attribute-based encryption systems is given in the full version of this paper. We now give our first formalization of the Lewko *et al.* construction [10] in AutoG&P. This scheme uses the three subgroup assumptions in composite order groups. The construction of [10] is given below.

1) $(PK, MSK) \leftarrow \mathsf{Setup}(\lambda, U)$: The setup algorithm chooses a bilinear group G of order $N = (p_1 p_2 p_3)$, where $p_1 p_2 p_3$ are three distinct primes. We let G_{p_i} denote the subgroup of order p_i in G. It then chooses random exponents $\alpha, a \in \mathbb{Z}_N$, and a random group element $g \in G_{p_1}$. For each attribute $i \in U$, it chooses a random value $s_i \in \mathbb{Z}_N$. The public parameters PK are $N, g, g^a, e(g, g)^\alpha, T_i = g^{s_i}\, \forall i$. The master secret key MSK is α and a generator X_3 of g_{p_3}.
2) $SK \leftarrow \mathsf{KeyGen}(MSK, S, PK)$: The key generation algorithm chooses a random $t \in \mathbb{Z}_N$, and random elements $R_0, R'_0, R_i \in G_{p_3}$. Compute $K = g^\alpha g^{at} R_0,\ L = g^t R'_0,\ K_i = T_i^t R_i\, \forall i \in S$. Set $SK = (K, L, K_i\, \forall i \in S)$.
3) $CT \leftarrow \mathsf{Encrypt}(A, \rho, PK, M)$: A is an $l \times n$ matrix and ρ is a map from each row A_x of A to an attribute $\rho(x)$. The encryption algorithm chooses a random vector $v \in \mathbb{Z}_N$, denoted by $v = (s, v_2, \cdots, v_n)$. For each row A_x of A, it chooses a random $r_x \in \mathbb{Z}_N$. The ciphertext is ((A, ρ) is implicitly included in the ciphertext): $C = Me(g, g^{\alpha s}), C' = g^s, C_x = g^{aA_x \cdot v} T_{\rho(x)}^{-r_x}, D_x = g^{r_x} \forall\, x$.
4) $M \leftarrow \mathsf{Decrypt}(CT, SK, PK)$: The decryption algorithm computes constants $\omega_x \in \mathbb{Z}_N$ such that $\Sigma_{\rho(x) \in S} \omega_x A_x = (1, 0, \cdots, 0)$. It then computes: $e(C', K)/\Pi_{\rho(x) \in S}(e(C_x, L) e(D_x, K_{\rho(x)}))^{\omega_x} = e(g, g)^{\alpha s}$.

Definition 3.1: (Extended Logic Rules)

$$\mathsf{Ext}(f(i_1, i_2, \cdots, i_n)) :$$
$$\frac{[G : ev\,(\mathsf{read}(a, f(i_1, i_2, \cdots, i_n)) \neq \mathsf{read}(b, f(i_1, i_2, \cdots, i_n)))] \preceq \epsilon}{[G : ev(a \neq b)] \preceq \epsilon}$$

$$\mathsf{R\text{-}over\text{-}W}(f(i_1, i_2, \ldots, i_n), f'(i'_1, i'_2, \ldots, i'_n)) :$$
$$\frac{\begin{array}{c}[G : ev\,(f(i_1, i_2, \ldots, i_n) = f(i'_1, i'_2, \ldots, i'_n))\wedge \\ \mathsf{read}(\ldots \mathsf{read}(\mathsf{read}(a[i_1]), i_2), \ldots, i_n)] \leq \epsilon_1 \\ [G : ev(f(i_1, i_2, \ldots, i_n) \neq f(i'_1, i'_2, \ldots, i'_n))\wedge \\ \mathsf{read}(\ldots \mathsf{read}(\mathsf{read}(a[i'_1]), i'_2), \ldots, i'_n)] \leq \epsilon_2\end{array}}{\begin{array}{c}[G : ev[\mathsf{read}(\ldots \mathsf{read}(\mathsf{read}(\mathsf{write}(\ldots \mathsf{write}(\mathsf{write}(a[i_1]), i_2), \ldots, i_n), v), \\ (a[i'_1]), i'_2), \ldots, i'_n)] \leq \epsilon_1 + \epsilon_2\end{array}}$$

$$\mathsf{W\text{-}Elim}(f(i_1, i_2, \ldots, i_n)) :$$
$$\frac{\begin{array}{c}[G : ev((f(i_1, i_2, \ldots, i_n \in \mathcal{I}) \wedge (a =_{\mathcal{I}} b))] \leq \epsilon_1 \\ [G : ev(f(i_1, i_2, \ldots, i_n \notin \mathcal{I}) \wedge (\mathsf{read}(\ldots \mathsf{read}(\mathsf{read}(b[i_1]), i_2, \ldots, i_n)) = v)\wedge \\ (a =_{\mathcal{I}} b))] \leq \epsilon_2\end{array}}{[G : ev\,(\mathsf{write}(\ldots \mathsf{write}(\mathsf{write}(a[i_1]), i_2), \ldots, i_n), v) =_{\mathcal{I}} b] \leq \epsilon_1 + \epsilon_2}$$

$$\mathsf{Partial\text{-}Eq}(f(i_1, i_2, \ldots, i_n)) :$$
$$\frac{\begin{array}{c}[G : ev(((f(i_1, i_2, \ldots, i_n) \notin \mathcal{I}) \wedge (a =_{\mathcal{I}} b)\wedge \\ \mathsf{read}(\ldots \mathsf{read}(\mathsf{read}(a[i_1]), i_2), \ldots, i_n) = \mathsf{read}(\ldots \mathsf{read}(\mathsf{read}(b[i_1]), i_2), \ldots, i_n))] \leq \epsilon_1 \\ [G : ev((f(i_1, i_2, \ldots, i_n) \in \mathcal{I}) \wedge (a =_{\mathcal{I}}))] \leq \epsilon_2\end{array}}{[G : ev(a =_{\mathcal{I}} b)] \leq \epsilon_1 + \epsilon_2}$$

$$\mathsf{Trans} :$$
$$\frac{[G : ev\,((a =_{\mathcal{I}} b) \wedge (a =_{\mathcal{I}'} c) \wedge (b =_{\mathcal{I} \cup \mathcal{I}'} c))] \preceq \epsilon}{[G : ev\,((a =_{\mathcal{I}} b) \wedge (a =_{\mathcal{I}'} c))] \preceq \epsilon} \boxed{\mathcal{I} \neq \emptyset \,\text{and}\, \mathcal{I}' \neq \emptyset}$$

$$\mathsf{Subst}(f(i_1, i_2, \ldots, i_n)) :$$
$$\frac{[G : ev\,(b, f(i_1, i_2, \ldots, i_n))] \preceq \epsilon}{[G : ev\,(a, f(i_1, i_2, \ldots, i_n))] \preceq \epsilon} \boxed{|a| \geq |b|}$$

$$\mathsf{Symm} :$$
$$\frac{[G : ev\,(b =_{\mathcal{I}} a)]}{[G : ev\,(a =_{\mathcal{I}} b)]} \boxed{|a| < |b|}$$

Fig. 1: Our Logic Rules

The M can be recovered by doing $C/e(g,g)^{\alpha s}$.

There are two structures used in the [10] scheme: semi-functional ciphertexts and semi-functional keys. These are not used in the real system, but are an artifact of the proof.

1) Semi-functional ciphertext: Let g_2 be a generator of G_{p_2}, and c be a random exponent modulo N. We also choose random values $z_i \in \mathbb{Z}_N$ associated to attributes, random values $\gamma_x \in \mathbb{Z}_N$ associated to matrix rows x, and a random vector $u \in \mathbb{Z}_N^n$. Then a semi-functional ciphertext C' is defined as: $C' = g^s g_2^c, C_x = g^{aA_x \cdot v} T_{\rho(x)}^{-r_x} g_2^{A_x \cdot u + \gamma_x z_{\rho(x)}}, D_x = g^{r_x} g_2^{-\gamma_x}\ \forall x$.
2) Semi-functional key: A semi-functional key will take on one of two forms. A semi-functional key of type I, also called a "nominally semi-functional" key is formed as follows. Exponents $t, d, b \in \mathbb{Z}_N$ and elements $R_0, R'_0, R_i \in G_{p_3}$ are chosen randomly. The key is set as:

$$K = g^{\alpha} g^{at} R_0 g_2^d,\ L = g^t R'_0 g_2^b, K_i = T_i^t R_i g_2^{bz_i}\ \forall i \in S.$$

A semi-functional key of type 2 is formed without the terms g_2^b and $g_2^{bz_i}$:

$$K = g^\alpha g^{at} R_0 g_2^d,\ L = g^t R_0',\ K_i = T_i^t R_i \forall i \in S.$$

We follow the proof as presented in the paper. At a high-level, the proof works in a series of hybrids that reduce a cpabe-adversary to the subgroup decision 1 adversary. The reduction first transforms a real game, $Game_{Real}$, with normal ciphertexts and normal keys to $Game_0$, where all the keys will be normal, but the ciphertext will be semi-functional. Then $Game_0$ is further reduced to a game $Game_{k,1}$, where the challenge ciphertext becomes semi-functional, the first $k-1$ keys become semi-functional, the k^{th} key becomes nominally semi-functional, and the remaining keys are normal. $Game_{k,1}$ is further transformed into a game $Game_{k,2}$ where the challenge ciphertext is semi-functional, the first k keys are semi-functional, and the remaining keys are normal. The final reduction is between $Game_{k,2}$ to a game $Game_{final}$, in which all the keys are semi-functional, and the challenge ciphertext is semi-functional, and hence none of the keys are useful in decrypting anything, and hence the adversary's advantage is 0. Let us call the reduction $Game_{Real} \rightarrow Game_0$ as G_0, the reduction $Game_0 \rightarrow Game_{k,1}$ as G_1, the reduction $Game_{k,1} \rightarrow Game_{k,2}$ as G_2, and the reduction $Game_{k,1} \rightarrow Game_{final}$ as G_3 respectively. We now give the formalizations in AutoG&P for the hybrid games. The superscripts on the games denote the initial game in the proof search procedure. Due to space constraints, we only give the game G_0^0 here, we give the proof search procedure, and intermediate hybrids in the full version.

Game G_0^0

```
1: $(N, g', g_t, e, g, X_3, T) \leftarrow B_1()$ with
    1.1: KeyGen(Gen){
        1.1.1: $p_1, p_2, p_3 \xleftarrow{\$} \mathbb{F}_p$;
        1.1.2: let $N = p_1 p_2 p_3; X_3 \xleftarrow{R} G_{p_3}; b \xleftarrow{\$} Bool$;
        1.1.3: return $(N, g', g_{t,}, e, g, X_3;$
                $(b?(T \xleftarrow{\$} G_{p_1 p_2}) : (T \xleftarrow{\$} G_{p_1})))$
    1.2 };
2: $a, \alpha \xleftarrow{\$} \mathbb{F}_p$;
3: let $pk = (N, g, g^a, e(g,g)^\alpha)$;
4: $\mathcal{S}_1[1..q_1] \xleftarrow{\$} \mathbb{F}_p$;
5: $(K', L, K[1..q_1]) \leftarrow A1(pk, \mathcal{S}_1)$ with;
    5.1: $B2(pk, \alpha, X_3)${
    5.2: let $t \xleftarrow{\$} \mathbb{F}_p$;
    5.3: let $R_0, R_0' \in G_{p_3}; R'[1..q_1] \xleftarrow{\$} \mathbb{F}_q$;
         let$R = [g^{R'[1]}, \cdots, g^{R'[q_1]}]$;
    5.4: let $K[1..q_1] \in G_{p_1 p_3}$;
    5.5: $K' = g^{\alpha + at} \cdot R_0, L = g^t R_0'$;
    5.6: for $i = 1$ to $q_1$
        5.6.1: write$(K, i, g^{S_1[i]+t} \cdot R[i])$;
        5.6.2: return $(K', L, K)$;
6: };
7: $(C, C', C'', D) \leftarrow A1(m_0, m_1, A^*, \rho, l, n)$ with
    7.1: $B3(m_0, m_1, A^*, \rho, l, n)${
        7.1.1: $b \xleftarrow{\$} Bool$; let$\mathcal{R}[1..l] \xleftarrow{\$} \mathbb{Z}_N$;
        7.1.2: $C[1..l] \xleftarrow{\$} \mathbb{F}_p$; let $C' = [g^{C[1]}, \cdots, g^{C[l]}]$;
        7.1.3: let $C = (m_b e(g^{\alpha, T}))$, let $C'' = T$;
        7.1.4: $V[1..n] \xleftarrow{\$} \mathbb{Z}_N$;
               let $V' = [1, g^{V[2]}, \cdots, g^{V[n]}]$;
        7.1.5: for $i = 1$ to $l${
        7.1.6: write$(C, i, T^{a \cdot A^*[i] \cdot V'} T^{-\mathcal{R}[i] \cdot s_{\rho(x)}})$;
        7.1.7: write$(D, i, T^{\mathcal{R}[i]})$;
        7.1.8: return $(C, C', C'', D)$;
8: };};
9: $S_1[q_1 + 1..q] \xleftarrow{\$} \mathbb{F}_p$;
10: $(K', L, K[q_1 + 1..q]) \leftarrow A1(pk, S_1)$ with;
    10.1: $B2(pk, \alpha, X_3)${
    10.2: let $R_0, R_0'; R'[q_1 + 1, \cdots, q] \xleftarrow{\$} \mathbb{F}_q$;
          let $R = [g^{R'[q_1+1]}, \cdots, g^{R'[q]}]$.
    10.3 let $K[q_1 + 1..q] \in G_{p_1 p_3}$;
    10.4: $K' = g^{\alpha + at} \cdot R_0, L = g^t R_0'$;
    10.5: for $i = q_1 + 1$ to $q$;
        10.5.1: guard$(K \notin A^*)$
        10.5.2: write$(K, i, g^{S_1[i]+t} \cdot R[i])$;
        10.5.3: return $(K', L, K)$;
11: } : $b = b'$
```

We assume that $g \in G_{p_1}$, $g_2 \in G_{p_2}$, $g_3 \in G_{p_3}$, $g' \in G$ and $g_t \in G_T$ are the respective generators, and $e : G^2 \rightarrow G_T$ is a bilinear map. $|\mathbb{G}_T| = |\mathbb{G}| = p_1 p_2 p_3$. In line 1, the subgroup assumption 1 adversary, B is given the public parameters of the system, the generator g of the subgroup G_{p_1}, generator of subgroup $X_3 \in G_{p_3}$, and its challenge, T, among other things. In Line 5, A requests its secret keys, which B responds correctly to. In Line 7, A sends its messages m_0, m_1, access structure A^*, mapping between attributes and rows, ρ, and requests its challenge ciphertext, in response to which B generates and returns the correct ciphertext tuple (C, C', C'', D). A can again query for private keys in Line 10, with the restriction that the private keys cannot satisfy A^*, which is enforced by the guard. In Line 11, B outputs A's guess as its own guess. We note that there are some syntactic issues in AutoG&P (e.g., we cannot directly sample elements from a group G, but need to sample $x \xleftarrow{\$} \mathbb{Z}_q$, and then do g^x, where $g \in G$, etc.), but we omit these from our game formalization in some places, for readability. We now give the Waters CP-ABE scheme based on the q-BDHE assumption.

V. Waters' Scheme based on q-BDHE assumption

This scheme is based on the Decisional q-Bilinear Diffie Hellman Exponent Assumption, which is considered stronger than the simple Decisional Bilinear Diffie Hellman (DBDH) assumption, but weaker than the q-Parallel Bilinear Diffie Hellman Exponent Assumption (q-PBDHE). We describe synthesizing the game hybrids and proof search procedure in the full version. The main challenge in the proof is to find a way to embed the adversary's challenge parameters into the public parameters generated by the simulator. The scheme below is proven secure under the restriction that an attribute appears in only one row in the access structure, i.e, the attribute-to-row mapping function $\rho(\cdot)$ is injective, although Waters [9] informally describe how the scheme can be extended to the situation where a single attribute is used in multiple rows, by adding an unique identifier to each instance, e.g., if attribute A_1 is used in multiple rows, one could just label each instance separately, e.g., $A_1 : 1, A_1 : 2$, etc. In the non-injective variant, the size of users' secret key grows to $|S| \cdot k_{\max}$, where $k_{\max}$ is maximum number of instances of an attribute. In the simple, injective variant, the size of users' secret key is $|S|$, where S is the set of attributes the user possesses.

A. Proof

At a high level, the proof aims to construct a simulator B that can solve the q-BDHE problem in polynomial time with non-negligible advantage, by interacting with an adversary A, which has a non-negligible advantage of breaking the security of the CP-ABE scheme in the selective security model. Let us assume the existence of a CP-ABE adversary A, that chooses a challenge access structure (M^*, ρ^*), where the size of M^* is $l^* \times n^*$, where $n^* \leq q$. We also assume that the total number of attribute sets, and the number of attributes in each set is upper-bounded by the max. number of system attributes, U. A has non-negligible advantage of $\epsilon = Adv_A$ in the CP-ABE selective security game. We also assume that the challenger for B is given a group $\mathbb{G}$ of prime oder p, $g \in \mathbb{G}$, the max. number of attributes in the system, U, and it sets B's challenge in accordance with these parameters. The proof was given for symmetric, type I groups; we follow the same.

The original (manual) proof of Waters had just one set of attributes S in phase 1, phase 2, and the challenge-response phase, presumably for simplicity, and uncluttered presentation. The idea was that the two phases, and the challenge-response step will be repeated multiple times, up until the appropriate set bounds. We could, in principle, do the same, but we choose to give the more general version, with sets $S_1, \cdots, S_{q_1}$, and $s_{q_1+1,\cdots,q}$, and the corresponding sets of secret keys in all three steps. This makes synthesizing the simulator tricky, introduces quite a bit of new notation, and the proof is more complicated (notationally, and in terms of reasoning too) than what it would have been in the single set case. . We describe how to automate the proof of Waters' scheme based on the q-PBDHE assumption, which is stronger than the q-BDHE assumption in the full version of the paper.

VI. Experiments

We have implemented the full automation of the proofs of the Lewko *et al.* scheme [10], and the Waters' scheme [9] in AutoG&P, the results are tabulated in Table I.[1]. In all cases, the proof is discovered semi-automatically, with the lines of code including the manual hand-tuning steps. Our experiments were run on an Intel core $i3-7100\text{T}$ (Dual Core, 3MB, 4T, 3.4GHz) running Ubuntu 16.04. The Lewko *et al.* scheme proof generation time is the sum of the proof-generation time for 4 different intermediate hybrid games (Game G_0, Game G_1, Game G_2, and Game G_3). Waters' scheme1 is the scheme based on the qBDHE assumption, and scheme2 is the scheme based on the qPBDHE assumption.

[1]https://github.com/sigcrypto/sigs-autognp

TABLE I
EXPERIMENTAL RESULTS

Proof of Scheme	Lines of Code	Time (ms)
Lewko *et al.* scheme [10]	343	238
Waters' scheme1 [9]	105	79
Waters' scheme2 [9]	121	97

VII. Future Work

An interesting direction to pursue is to explore how to extend AutoG&P to support multilinear or k-linear pairings, as opposed to just bilinear pairings. A natural step would also be to explore if we can automate proofs of attribute-based signatures schemes [**?**], and proofs of other, more generalized forms of attribute-based encryption such as functional encryption or predicate encryption. In applications where independent proof verification by other tools is desirable, one could explore if proofs output by AutoG&P can still be exported into Easy-Crypt or Coq (only initial version had support) for verification.

References

[1] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin, "Automated analysis of diffie-hellman protocols and advanced security properties," in *25th IEEE Computer Security Foundations Symposium, CSF*, 2012, pp. 78–94.

[2] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub, "Easycrypt: A tutorial," in *Foundations of Security Analysis and Design FOSAD*, 2013, pp. 146–166.

[3] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *2006 IEEE Symposium on Security and Privacy (S&P)*, 2006, pp. 140–154.

[4] I. labs, "The Coq proof assistant," https://coq.inria.fr/.

[5] A. Petcher and G. Morrisett, "The foundational cryptography framework," in *Principles of Security and Trust - 4th International Conference, POST, Proceedings*, 2015, pp. 53–72.

[6] J. A. Akinyele, C. Garman, and S. Hohenberger, "Automating fast and secure translations from type-i to type-iii pairing schemes," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, 2015, pp. 1370–1381.

[7] G. Barthe, B. Grégoire, and B. Schmidt, "Automated proofs of pairing-based cryptography," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security, CCS*, 2015, pp. 1156–1168.

[8] G. Eswaraiah, R. Vishwanathan, and D. Nedza, "Automated proofs of signatures using bilinear pairings," in *16th Annual Conference on Privacy, Security and Trust, PST 2018*, 2018, pp. 1–10.

[9] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, 2011, pp. 53–70.

[10] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, 2010, pp. 62–91.