Automated Proofs of Signatures using Bilinear Pairings

Guruprasad Eswaraiah, Roopa Vishwanathan *New Mexico State University*
{guru,roopav}@nmsu.edu
Douglas Nedza *SUNY Polytechnic*
nedzad@sunyit.edu

***Abstract*—In this paper, we extend an automated proof-generation tool, AutoG&P with new axioms and formalizations to support composite data types and q-type assumptions, which in turn can be used to automate pairing-based signature schemes. AutoG&P due to Barthe *et al.* was designed as a tool to automate proofs of cryptographic primitives based on bilinear pairings in the standard model, but the initial version only supported a limited set of data types, limited pairing-based assumptions, and only provided automated proofs for *encryption* schemes, notably the Boneh-Boyen identity-based encryption scheme. As examples of our extensions, we provide automated proofs for the Boneh-Boyen pairing-based signature schemes under the well-known and widely-used notion of signature security: existential unforgeability under chosen message attacks in the standard model, and the Boneh-Boyen-Shacham group signature scheme, under standard notions of group signature security: anonymity and traceability.**

***Index Terms*—pairing-based cryptography, bilinear pairings, signatures**

I. Introduction

Cryptographic protocols, which form the backbone of several communication networks and infrastructure, are proved secure in well-established formal frameworks, and designers of such protocols and primitives spend a lot of time and effort on carefully defining security properties, and proving that their designs possess these properties. As cryptographic protocols increase in complexity, consequently, so do their security definitions and proofs, and there is a real need for automated tools, such as proof assistants, to help construct and verify cryptographic proofs. In recent years, there has been a concerted effort to create automated toolsets that aid in cryptographic proof construction and verification, such as EasyCrypt due to Barthe *et al.* [1] and CryptoVerif due to Blanchet [2]. These tools help in creating and verifying proofs in the game-based reductionist style that cryptographers are familiar with, and also take into account implementation details and standards recommendations.

Halevi, in his influential paper in 2005 [3], first mooted the idea of creating automated tools for cryptographic proof verification. His vision was one of automated 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 position 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 [4]. 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.

Initial efforts in this area were directed towards using machine-checked automated frameworks from the *symbolic model* of cryptography due to Dolev and Yao [4], which did yield some interesting results [5], [6], [7], [8], [9], but this approach was quickly found to be inadequate, and researchers have recognized the need for building automated proof assistants in the *computational model*, in which almost all cryptographic protocols are defined and proven secure. The symbolic model of Dolev and Yao, while seemingly attractive because of its simplicity, is one where all cryptographic primitives are assumed to be black boxes that the adversary cannot tamper with. This model lends itself easily to the construction of automated verification tools, however the restricted capabilities of the adversary make the model quite unrealistic, and proofs in the symbolic model are generally considered weaker than the computational model.

In the computational model, adversaries are treated as probabilistic polynomial time Turing machines, messages are treated as bitstrings, and the adversary has the ability to tamper with the cryptographic primitives, but proofs in this model are harder to write and reason about compared to the easier symbolic model. Nevertheless, the security guarantees offered by the computational model are stronger than that offered by the symbolic model. As a bridge step between the two models, there were efforts to show that the weaker security guarantees offered by the symbolic model can be "transferred" to the computational model using principles of *computational soundness* [10], but this approach has revealed flaws in reasoning, and it has been shown [11] that proofs in the symbolic model do not necessarily always translate into proofs in the computational model, at least not without

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. 978-1-5386-7493-2/18/$31.00 ©2018 IEEE

additional, at times questionable assumptions and hypotheses, the main assumption being that a computational soundness result exists for every result whose security proof we want to translate from the symbolic to computational models. The more worrying issue is that computational soundness results make unrealistic use-case assumptions which may be hard to enforce, or may even be unenforceable in practice [11].

In response to this, recently researchers have proposed automated tools that work in the computational model of cryptography, such as EasyCrypt [1], CryptoVerif [2], CertiCrypt [12], ZooCrypt [13], and the more general-purpose proof assistant Coq [14] augmented with the Foundational Cryptography Framework library (FCF) [15]. On another note, an interesting observation is that most research into automated proof generation and verification, both in the symbolic and computational models, has focused, rather disproportionately on automating proofs of cryptographic *protocols*, not *primitives*. Of the ones mentioned here, 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 [16], [17], [18], and the standard model [19]. In this paper we focus on standard model tools, specifically, the AutoG&P tool of Barthe *et al.* The work of Barthe *et al.* has two main contributions: 1) introducing a formal logic consisting of a set of rules that model standard proof steps (series of reductions, transformations, sampling distributions, among others), and providing a proof of soundness of their formal logic. 2) the second contribution is an automated tool, AutoG&P which implements the rules of the formal logic, and is able to reason about, at a basic level, the security of primitives based on bilinear pairing assumptions. As case studies, the authors showed how to apply AutoG&P to obtain proofs for the CPA security of the Boneh-Boyen identity-based encryption scheme in the selective-ID model [20], and the CPA security of Waters' dual system identity-based encryption system [21].

Our work was motivated by the relatively sparse work done in the area of automating proofs of cryptographic primitives, and more so in advanced areas such as pairing-based cryptography. Pairing-based cryptography was introduced in the early 2000's and has formed the basis of numerous encryption and signature schemes [20], [22], which have in turn formed the foundations of new families of cryptosystems, such as identity-based cryptosystems, non-interactive zero knowledge proofs, and attribute-based cryptography, to name a few.

A. Our Contributions

Our two main contributions are: 1) to extend the type grammar and semantics of AutoG&P to axiomatize composite data types such as arrays in security games, by using ideas from automated reasoning and symbolic computation, which helps expand the scope of AutoG&P to support q-type assumptions which are used as building blocks in the construction of several pairing-based primitives and protocols. 2) to extend the logic of AutoG&P to support the widely-used notion of signature security: existential unforgeability under chosen message attacks, and support automated proofs of certain signature schemes that use bilinear pairings. The initial version of AutoG&P was able to reason about chosen-plaintext and chosen-ciphertext indistinguishability in the context of encryption schemes that are built upon pairing assumptions, but did not support notions of signature security. Using these building blocks, we provide automated proof search procedures for the Boneh-Boyen signature scheme [23], and the anonymity, traceability properties of the Boneh-Boyen-Shacham group signature scheme [24] using AutoG&P.

II. Related Work

CryptoVerif, due to Blanchet [2], was the first automated prover that can prove statements of cryptographic protocols in the computational model. One of the main contributions of CryptoVerif was that it laid down an automation framework and template for most or all work in this area that followed it, i.e., discharging proof obligations in a sequence of games, with transformations between them, starting from the higher-order security guarantees, and reducing them, in a step-by-step process to fundamental mathematical assumptions. CryptoVerif has built-in transformations, which can apply reductions to games in a fully automated way, along with an interactive mode meant for situations in which the prover would like to participate in creating the proof. CryptoVerif wasn't designed for, and as such hasn't been used for pairing-based cryptographic schemes.

CertiCrypt [12] is another tool that has been used to verify the security of cryptographic primitives using the general-purpose Coq proof assistant [14]. CertiCrypt has been used to generate proofs of useful crypto primitives such as zero-knowledge proofs [25], OAEP [26], identity-based encryption [27], and Full Domain Hashes (FDH) [28]. Easycrypt due to Barthe *et al.* [1] is an automated tool that can produce machine-checked proofs from proof sketches (of protocols and a few primitives). EasyCrypt uses the game-playing technique, similar to CryptoVerif and CertiCrypt, and the proofs output by (initial version of) EasyCrypt could be verified by general-purpose proof assistants such as Coq [14]. Probabilistic relational Hoare Logic is used for proving transitions between intermediate games, and for deriving claims about event probabilities in the games; and a simple tactics language is used for proving judgments. As example applications, the authors show how to generate and verify proofs of the Cramer-Shoup cryptosystem and the Hashed El-Gamal cryptosystem using EasyCrypt. Prior work due to Barthe *et al.* also resulted in the development of ZooCrypt [13], which provided a way to analyze and prove chosen plaintext and chosen ciphertext security of padding-based encryption schemes in the random oracle model. Computational Indistinguishability Logic (CIL) [29] is a general-purpose logic that does not work in the game-based reductionist model of cryptographic proofs, and does not have support for automation. Barthe *et al.* also propose automated tools for the special-case of structure-preserving signatures in the generic group model [30].

Although they represent important, pioneering work in the area of proof automation, CryptoVerif, CertiCrypt, EasyCrypt, ZooCrypt were not designed for automating or verifying proofs that use pairing-based cryptography. Akinyele *et al.* [31] proposed new automated tools for pairing-based encryption and signature schemes; their tools are meant for automating the construction of cryptographic protocols (not proofs), whereas our goal is the automation of cryptographic proofs in the bilinear pairings setup.

A. *AutoG&P*

AutoG&P [19] is a tool that can analyze the security of pairing-based cryptographic primitives in the standard model, and implements a set of high-level logic rules for performing reductions, transforming games, and applying inferences based on computational or decisional assumptions. AutoG&P has its own syntax for writing expressions, and has a well-defined grammar for writing games (or security experiments), which could possibly include an adversary being provided access to encryption/decryption oracles. It also provides a syntax for writing probability judgments, and bounding the advantage of adversaries with the hardness of breaking some fundamental decisional or computational pairing-based assumption. The core rules of AutoG&P consist of operations that are used in proving equivalence between games such as transformation rules, reduction rules, random sampling rules and probability judgment rules.

III. New Logic Rules

A fundamental challenge we face is how to reason about and represent composite data types such as arrays and vectors in AutoG&P. Since AutoG&P does not support indexed variables, we cannot directly formalize games and proofs involving indexed variables, such as the q-SDH assumption in AutoG&P. This presents us with some challenging questions: How does one reason about composite data types, such as arrays, in the context of an automated theorem prover? How does one formalize the properties and operations of composite data types? What does it mean for an array implementation to be secure?[1] There is significant work, both theoretical [32], [33], [34] and experimental [35], [36], [37], in the area of building theorem provers with the ability to reason about, and solve formulas that involve arrays. Broadly, two kinds of array theories have been studied in logic and symbolic computation, which can be applied to areas such as formal verification of software, and design of theorem provers and proof assistants: *extensional* array theories, and *non-extensional* array theories. Non-extensional theories are axiomatized by the `read` and `read-over-write` axioms; extensional theories have the additional `equality` axiom which states that two arrays are equal if they store the same element at each index i. We give the definition of an extensional theory of arrays below.

[1]In this paper we focus on adding support for the array data type; in the future we plan to investigate other abstract and algebraic data types.

Definition 3.1: (Extensional Theory of Arrays) [38], [32]: For all arrays a, b, indices i, j, and values v of a given type, an extensional theory of arrays is defined by the following axioms:

1) Read-over-Write: $((i = j) \to \text{read}(\text{write}(a, i, v), j) = v) \wedge ((i \neq j) \to \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j))$
2) Extensionality: $(\text{read}(a, i) = \text{read}(b, i)) \to (a = b)$

SMT solvers usually instantiate arrays using "lazy instantiation", wherein the array operations such as `read` and `write` are implemented by uninterpreted functions, using the McCarthy extensional theory of arrays in first-order logic [38], which is considered a classical result in array theory by the automated reasoning community. The McCarthy theory of arrays uses uninterpreted functions to formally represent state (e.g., of main memory) during execution of programs that use arrays. The basic theory is not expressive enough, since it only supports write operations at a single memory location, and not to a contiguous memory block, in a similar vein as C-language `memset` or `memcpy` functions. Since then, more advanced theories, which provide formalizations for an extended set of array operations, including `set` and `copy` have been proposed (e.g., [39]). Stump *et al.* [32] present an extensional theory of arrays that axiomatizes a range of operations in addition to read and write, such as transitivity, symmetry, etc. They provide a decision procedure for the quantifier-free fragment of the theory, and their theory can potentially be extended to multi-dimensional arrays. For formulating our rules, we use the extensional array theory as introduced by Stump *et al.* [32] as a foundation. Before presenting our new rules, we first give a definition that describes an algorithm for eliminating write expressions occurring in equations involving arrays.

Definition 3.2: (Partial equations [32]) Let a, b be arrays, let $I, \mathcal{I}$ be sets of an index data type, let V be a set of a value data type, let $v \in V$, and let p denote a unique index in an array. Then:

$$a =_{\mathcal{I}} b \Leftrightarrow_{\text{def}} \forall p : I.\, p \notin \mathcal{I} \to \mathsf{read}(a, p) = \mathsf{read}(b, p), \text{and hence}$$

$$\mathsf{write}(a, p, v) = b \Leftrightarrow (a =_{\{p\}} b \wedge \mathsf{read}(b, p) = v)$$

Formulas of the form $a =_{\mathcal{I}} b$ with $\mathcal{I} \notin \emptyset$ are called partial equations.

We now give our seven extended rules for AutoG&P: Ext, R-over-W, W-Elim, Partial-Eq, Trans, Subst, and Symm. In the rules, G is a game, ev is an event expression, and $[G : ev]$ describe a security experiment, $[G : ev] \preceq \epsilon$ is an expression that bounds the probability of event ev in game G by ϵ, and p is a position (index) in an array. For each rule, the goal below the line is rewritten to the subgoals above the line.

Definition 3.3: (Extended Logic Rules)

$$\mathsf{Ext}(p) : \frac{[G : ev\,(\mathsf{read}(a, p) \neq \mathsf{read}(b, p))] \preceq \epsilon}{[G : ev(a \neq b)] \preceq \epsilon}$$

$$\mathsf{R\text{-}over\text{-}W}(p, p') : \frac{\begin{array}{c}[G : ev\,((p = p') \wedge \mathsf{read}(a, p))] \preceq \epsilon_1 \\ [G : ev\,((p \neq p') \wedge (\mathsf{read}(a, p')))] \preceq \epsilon_2\end{array}}{[G : ev\,(\mathsf{read}(\mathsf{write}(a, p, v), p'))] \preceq \epsilon_1 + \epsilon_2}$$

$$\mathsf{Subst}(p) : \frac{[G : ev\,(b, p)] \preceq \epsilon}{[G : ev\,(a, p)] \preceq \epsilon} \; \boxed{|a| \geq |b|}$$

$$\mathsf{W\text{-}Elim}(p) : \frac{\begin{array}{r}[G : ev\,((p \in \mathcal{I}) \wedge (a =_{\mathcal{I}} b))] \preceq \epsilon_1 \\ [G : ev((p \notin \mathcal{I}) \wedge (\mathsf{read}(b,p) = v) \wedge (a =_{\mathcal{I}} b))] \preceq \epsilon_2\end{array}}{[G : ev\,(\mathsf{write}(a,p,v) =_{\mathcal{I}} b)] \preceq \epsilon_1 + \epsilon_2}$$

$$\mathsf{Partial\text{-}Eq}(p) : \frac{\begin{array}{r}[G : ev\,((p \notin \mathcal{I}) \wedge (a =_{\mathcal{I}} b) \wedge (\mathsf{read}(a,p) = \mathsf{read}(b,p)))] \preceq \epsilon_1 \\ [G : ev\,((p \in \mathcal{I}) \wedge (a =_{\mathcal{I}} b))] \preceq \epsilon_2\end{array}}{[G : ev\,(a =_{\mathcal{I}} b)] \preceq \epsilon_1 + \epsilon_2} \; \boxed{|a| \geq |b|; \mathcal{I} \notin \emptyset; \mathsf{read}(a,p) \text{ occurs in } [G : ev]}$$

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

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

The extensionality rule, Ext, formalizes that if two arrays are not equal in a security experiment, then the values they store at index p will be different. The read over write rule, R-over-W, writes a value v to index p in array a, then reads the value at index p'. If $p = p'$, then it returns v, else it reads and returns the result of $\mathsf{read}(a, p')$. The Partial-Eq rule formalizes the first point of Definition 3.2 and states that if two arrays a and b agree at every index, except those in the set $\mathcal{I}$, then for every $p \notin \mathcal{I}$, $\mathsf{read}(a, p)$ should be equal to $\mathsf{read}(b, p)$. The write with elimination rule, W-Elim, formalizes the second point of Definition 3.2, which eliminates write expressions, replacing them with reads. The Trans rule formalizes transitivity between three arrays a, b, c. The Subst rule replaces all occurrences of array a with array b in a game. The Symm rule states that if array a stores the same value as 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 $\mathcal{I}$.

Some rules must be restricted to avoid non-termination due to repeated and trivial applications of the rule, such as the Partial-Eq and Trans, which we enforce by adding side-conditions. In the Partial-eq rule, this is prevented by adding a (boxed) side-condition to the rule that prevents it from being applied if $\mathsf{read}(a, p)$ and $\mathsf{read}(b, p)$ are already known to be equal, or if p is already known to be an element of $\mathcal{I}$. Similarly, the side-condition in Trans prevents the rule from being applied if any two arrays are known to be equal at all indices, except those in a non-empty set. For now, we only work with single-dimensional arrays, so the above stand-alone theory suffices. In the future, when we work with two-dimensional arrays or multi-dimensional arrays (e.g., formalizing rules for monotone span programs or linear secret sharing scheme (LSSS) matrices), we will consider combining two array theories using theory combination methods [40], [41]. We state and prove a soundness theorem for our rules, which is given in the Appendix.

IV. Extensions to AutoG&P

We now describe the extensions we make to AutoG&P's grammar and type semantics for it to support array axioms. Our first extension is introducing a new parametric data type, denoted by "Arr", which can store a collection of variables of some of AutoG&P's existing data types: Boolean, bitstring, groups and (finite) fields. In Table I, we give the grammar of data types with our extension for arrays. In the table, Len denotes a finite set of length variables, GName is a finite set of group names.

TABLE I
Type grammar

Expression	Type
$\mathbb{B}$	Boolean
$\mathbb{BS}_l$	bitstrings of length $l \in \mathsf{Len}$
$\mathbb{G}_i$	cyclic group with $i \in \mathsf{GName}$
$\mathbb{F}_p$	finite field of order p
Arr_n	array of size n; $n \in \mathbb{Z}^+$ of the following types: $b_1, \cdots, b_n; b_i \in \mathbb{B}$ $bs_1, \cdots, bs_n; bs_i \in \mathbb{BS}_l$ $g_1, \cdots, g_n; g_i \in \mathbb{G}_i$ $f_1, \cdots, f_n; f_i \in \mathbb{F}_p$

AutoG&P provides a set of oracle and game grammars, which consist of oracle and game definitions and commands. Oracle and game definitions consist of a symbol, input parameters, a sequence of oracle/game commands and the value returned by the oracle or game. There are four possible game commands: let bindings (variable assignments), random sampling from a pre-defined distribution, an assertion which checks if a condition is true, and adversary calls. Oracle commands are defined in the same way as game commands, except one cannot have assertions within an oracle, and every oracle has a *guard*, which prevents the adversary from submitting, and obtaining answers to, invalid input, e.g., challenge ciphertexts. Adversaries are assumed to maintain state between calls to oracles. We introduce two new commands for an array read/write operation, based on the extensional theory of arrays. Table II shows the existing oracle and game commands with our additions. Some other extensions are described in the Appendix.

V. Boneh-Boyen Signature Schemes

The Boneh-Boyen signature scheme [23] is based on the q-SDH assumption, and has two variants, each corresponding to a different notion of security (please see Appendix for relevant assumption definitions). The *weak* scheme, which provides existential unforgeability under weak chosen message attacks,

TABLE II
ORACLE AND GAME COMMANDS

Action	Symbol	Type Grammar
Defining oracle	O	$o(x) = \{(\overrightarrow{oc}); \text{return}\, e\}$
Writing oracle command	oc	c; ordinary command $\mathsf{guard}(b)$; guard for expression $b \in \mathsf{Expr}_{\mathbb{B}}$
Writing game command	gc	c; ordinary command $\mathsf{assert}(ev)$; assertion for event expression $ev \in \mathsf{Expr}_{ev}$ $y \leftarrow A(x)$ with $\overrightarrow{O}$; adversary call with one or more oracles
Defining command	c	let $x = e$; let binding $x \xleftarrow{\$} t \backslash s$; sample x from distribution $t\backslash\{s\}$
Defining event expression	ev	e; event expression $\exists\, b_1, \cdots, b_n \cdot e$; there exist queries $\forall\, b_1, \cdots, b_n \cdot e$; for all queries
Query	b	$x \in Q_o$; x ranges over queries
Reading from an array	$a[p]$	$\mathsf{read}(a, p)$
Writing to an array	(a, p, val)	$write(a, p, \text{val})$

and the *strong* or the *full* scheme, which provides strong existential unforgeability against standard chosen message attacks. In the strong scheme, the existential unforgeability game requires that the adversary is not able to generate new (fake) signatures even on previously signed messages. Our goal is to use AutoG&P to construct proofs for each of these variants.

A. Weak Chosen Message Attacks

Let G_1, G_2 be prime-order cyclic groups where $|G_1| = |G_2| = p$, and let g_1 be a generator of G_1, and g_2 be a generator of G_2. Let there be an efficiently computable isomorphism ψ, such that $g_1 = \psi(g_2)$. Then the Boneh-Boyen weak signature scheme consists of the following three algorithms:

1) $(PK, SK) \leftarrow$ Key Generation(1^λ): Set $g_1 = \psi(g_2)$. Pick random $x \xleftarrow{\$} \mathbb{Z}_p^*$, compute $v \leftarrow g_2^x \in G_2$, $z \leftarrow e(g_1, g_2) \in G_T$. Set $PK = (g_1, g_2, v, z)$, and $SK = x$.
2) $\sigma \leftarrow$ Sign(x, m): Let $m \in \mathbb{Z}_p^*$. Compute signature $\sigma \leftarrow g_1^{\frac{1}{x+m}} \in G_1$; If $(x+m) = 0$, set $\sigma = 1$. Return σ.
3) ("accept","reject") $\leftarrow$ Verify(PK, m, σ): Check if $e(\sigma, v \cdot g_2^m) = z$. If yes, return accept, else reject.

We now give the weak chosen message attack game for the weak signature scheme. The scheme is considered existentially unforgeable under a weak chosen message attack, if no probabilistic polynomial time adversary can win the game with non-negligible probability.

1) Query phase: Adversary A sends the challenger a list of q messages: $m_1, \cdots, m_q$, where each $m_i \leftarrow \mathbb{Z}_p^*$.
2) Response phase: The challenger generates a keypair (PK, SK), and signs the adversary's messages: $\sigma_1 =$ Sign$(SK, m_1), \cdots, \sigma_q =$ Sign(SK, m_q). The challenger gives $(PK, \sigma_1, \cdots, \sigma_q)$ to the adversary.
3) Output phase: A outputs a pair (m, σ), and wins the game if:
   a) $m \notin \{m_1, \cdots, m_q\}$, and
   b) Verify$(PK, m, \sigma) =$ "accept".

The security of the weak signature scheme is based on the hardness of solving the q-SDH problem. The idea is to prove that if an adversary A can break the existential unforgeability of the weak scheme under a chosen message attack, then an adversary B can, by interacting with A, solve the q-SDH problem in polynomial time, with non-negligible probability.

Definition 5.1: A probabilistic polynomial-rime forger $A(t, q, \epsilon)$-weakly breaks a signature scheme if A runs in time at most t, makes at most q signature queries, and the advantage of A, Adv_A is at least ϵ, taken over the coin tosses of the adversary and the challenger. A signature scheme is (t, q, ϵ)-existentially unforgeable under a weak chosen message attack if no forger (t, q, ϵ)-weakly breaks it.

We now express the Boneh-Boyen weak EF-CMA game, and the q-SDH game using the logic rules of AutoG&P.

Game BB (G^{BB})

```
1: $m_1, \cdots, m_q \leftarrow \mathsf{A}_1()$;
2: $x \xleftarrow{\$} \mathbb{F}_p$, $z \leftarrow e(g_1, g_2)$; let $P = (g_1, g_2, g_2^x, z)$;
3: $M[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q$; $S'[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q$;
     let $S[1..q] \leftarrow \{g_1^{s'_1}, \cdots, g_1^{s'_q}\}$;
4: $(m^*, \sigma^*) \leftarrow \mathsf{A}_2(P, m_1, \cdots, m_q)$ with
5: $\mathsf{Sign}(m_1, \cdots, m_q) = \{$
6: for $(i = 1; i \leq q; i{+}{+})$
7: $\{$
8: $\sigma_i = g_1^{\frac{1}{x+m_i}}$;
9: guard$((x + m_i) \stackrel{?}{=} 0;\ \sigma_i \leftarrow 1)$;
10: $\mathtt{write}(M, i, m_i); \mathtt{write}(S, i, \sigma_i)$;
11: $\}$;
12: $\}$;
13: return $(m^* \neq \{m_1, \cdots, m_q\}, \sigma^*)$
```

Game q-SDH ($G^{q-\mathsf{SDH}}$)

```
1: $x \xleftarrow{\$} \mathbb{F}_p$;
2: $(c, g_1^{\frac{1}{x+c}}) \leftarrow \mathsf{B}(g_1, g_2, g_2^x, g_2^{x^2}, \cdots, g_2^{x^q})$;
```

Game G^{BB}: In line 1, the adversary chooses its signature queries (the Boneh-Boyen weak signature scheme requires that the adversary submits its signature query before seeing the public key). In lines 2,3 the challenger generates the secret key x, initializes the message/signature arrays, and computes the public parameters P, which include a bilinear map, $z \in G_T$. Note that per AutoG&P's (original) rules, one needs to normalize variables, i.e., instead of writing $u \in G_i$, we need to sample $x \xleftarrow{\$} \mathbb{F}_p$, and compute $u = g_i^x$, where g_i is the generator of $\mathbb{G}_i$. In line 4, the adversary is called, given access to the public parameters, and a signing oracle, and queries the challenger for signatures on the previously picked messages $m_1, \cdots, m_q$. In AutoG&P, adversaries maintain state between calls. The task of the adversary is to output a message/signature pair (m^*, σ^*) that passes verification with non-negligible probability. Line 9 checks if $(x + m_i) = 0$, in which case the signing oracle sets the signature to be 1. In the weak scheme, the adversary producing a message that corresponds to a previously obtained signature is not considered a forgery. If $\mathsf{A}_1, \mathsf{A}_2$ are the signature scheme adversaries, and B is the q-SDH adversary, then the judgment we need to prove is:

$$[G^{\mathsf{BB}}_{\mathsf{A}_1,\mathsf{A}_2} : \text{accept}]_{\mathsf{succ}} \preceq [G_B^{q-\mathsf{SDH}} : (c, g_1^{\frac{1}{x+c}}) \leftarrow \ B]_{\mathsf{succ}}$$

The above expression essentially says that the success probability in the weak scheme of all adversaries (i.e., an adversary returning a signature that passes verification), is upper-bounded by the success probability of B outputting a valid $(c, g_1^{\frac{1}{x+c}})$ pair in the q-SDH game.

The proof search involves automatically finding a series of transformations using the core rules of AutoG&P to turn a game for the weak EF-CMA scheme into a game for the q-SDH assumption. As shown in game G^{BB}, this involves first normalizing the random samplings of the challenger, and matching up sampling of variables in the EF-CMA game with the q-SDH game. Additionally, one is required to remove all "let" statements, e.g., setting the public parameters in Line 2 of the G^{BB} game, but we keep the let statements in the game here for readability. We also expand some of the statements involving arrays (e.g., $m_1, \cdots, m_q$, instead of M, and $M[1..q]$, instead of just M) for readability.

Next, we need to replace random variables generated and used by the challenger. In the G^{BB} game, this is an easy and intuitive step, since the only random variable sampled by the challenger is x. AutoG&P's rules replace $x \xleftarrow{\$} \mathbb{F}_p$ by $x' \xleftarrow{\$} \mathbb{F}_p$, then directly apply the q-SDH assumption, and obtain the desired bound. One facet of AutoG&P that makes things easy for us is that AutoG&P's conditional deducibility algorithm is tailored to deal with the expressions of the form $\Gamma \models \overrightarrow{x}, g_{i_1}^{f_1}, \cdots, g_{i_k}^{f_k} \vdash g_j^h$, where Γ is a set of axioms, $\overrightarrow{x}$ is a vector of variables of type $\mathbb{F}_p$, and f_i and h are polynomials. In our case, we only need to able to deal with monomials in the exponent for the q-SDH assumption. We now describe the simulator, B, built by the tool for the q-SDH assumption as applied to the weak signature scheme.

Simulator for q-SDH assumption in the weak scheme, B

```
1: $m_1, \cdots, m_q \leftarrow \mathsf{A}_1()$;
2: $x \xleftarrow{\$} \mathbb{F}_p$; let $P = (\nu, \gamma, \gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_q)$;
3: $M[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q$; $S'[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q$;
       let $S[1..q] \leftarrow \{g_1^{s'_1}, \cdots, g_1^{s'_q}\}$;
4: $(m^*, \sigma^*) \leftarrow \mathsf{A}_2(P, m_1, \cdots, m_q)$ with
5: Sign$(m_1, \cdots, m_q) = \{$
6: for $(i = 1; i \leq q; i++)$
7: $\{$
8: $\sigma_i = \gamma^{\frac{1}{x+m_i}}$;
9: guard$((x + m_i) \stackrel{?}{=} 0;\ \sigma_i = 1)$;
10: write$(M, i, m_i)$; write$(S, i, \sigma_i)$;
11: $\}$;
12: $\}$;
13: return $((\hat{e}, (\sigma^*, \gamma_1 * \gamma_0^{m^*}) \stackrel{?}{=} \nu)$
       && $(m^* \neq \{m_1, \cdots, m_q\}))$
```

Simulator B: In Line 1, the adversary chooses its challenge messages, $m_1, \cdots, m_q$. In Line 2, the challenger runs the key generation algorithm and generates a public/secret keypair, $SK = x; x \xleftarrow{\$} \mathbb{Z}_p^*$, and $PK = (\nu = e(g_1, g_2), \gamma = g_1, \gamma_0 = g_2, \gamma_1 = g_2^x, \gamma_2 = g_2^{x^2}, \cdots, \gamma_q = g_2^{x^q})$ (the proof search later involves renaming random variables in the game G^{BB} to match up with the simulator). In Line 3, the challenger initializes the message and signature arrays. In Line 4, the adversary is called, is given the public parameters, and chooses a set of messages $m_1, \cdots, m_q$; $m_i \in \mathbb{Z}_p^*$. The adversary is also given access to a signing oracle. Line 5-10 describe the actions of the oracle, which signs messages of the adversary's choice. The oracle has a guard, which returns $\sigma_i = 1$ in case $(x + m_i) = 0$, for some m_i. Finally, the adversary returns a message/signature pair (m^*, σ^*). In Line 13, we check if σ is valid, and $m^* \notin (m_1, \cdots, m_q)$.

B. Strong Existential Unforgeability

Let G_1, G_2 be prime-order cyclic groups where $|G_1| = |G_2| = p$, and let g_1 be a generator of G_1, and g_2 be a generator of G_2. Then the Boneh-Boyen strong signature scheme consists of the following three algorithms:

1) $(PK, SK) \leftarrow$ Key Generation(1^λ): Set $g_1 = \psi(g_2)$. Pick $x, y \xleftarrow{\$} \mathbb{Z}_p^*$, and compute $u \leftarrow g_2^x \in G_2$, $v \leftarrow g_2^y \in G_2$, and the bilinear map $z \leftarrow e(g_1, g_2) \in G_T$. Set $PK = (g_1, g_2, u, v, z)$, and $SK = (x, y)$.
2) $(\sigma, r) \leftarrow$ Sign(SK, m): Let $m \in \mathbb{Z}_p^*$. Pick $r \xleftarrow{\$} \mathbb{Z}_p^*$, and compute $\sigma \leftarrow g_1^{1/(x+m+yr)} \in G_1$. If $(x + m + yr) = 0$, try again with a different r. Return (σ, r).
3) ("accept","reject") $\leftarrow$ Verify(PK, m, σ, r): If $e(\sigma, u \cdot g_2^m \cdot v^r) = z$, return accept, else reject.

The notion of strong existential unforgeability under chosen message attack is defined by the following game between an adversary, A, and a challenger. We consider the static (non-adaptive) adversary version of the Boneh-Boyen strong signature scheme. The scheme is said to be strongly existentially unforgeable under a standard chosen message attack, if no probabilistic polynomial time adversary can win this game with non-negligible probability.

1) Setup phase: The challenger generates a public/secret keypair, (PK, SK), and gives PK to the adversary.
2) Query phase: A requests signatures on $m_1, \cdots, m_q$ messages of its choice. The challenger returns $\sigma_1, \cdots, \sigma_q$.
3) Challenge – response phase: A outputs a pair (m, σ), and wins the game if:
   a) $(m, \sigma) \notin \{(m_1, \sigma_1), \cdots, (m_q, \sigma_q)\}$, and
   b) $\mathsf{Verify}(PK, m, \sigma) =$ "accept".

Definition 5.2: A probabilistic polynomial-time forger $A(t, q, \epsilon)$-breaks a signature scheme if A runs in time at most t, makes at most q queries, and Adv_A is at least ϵ. A signature scheme is (t, q, ϵ)-existentially unforgeable under a chosen message attack if no forger (t, q, ϵ)-breaks it.

We now express the EF-CMA game of the strong scheme in the logic of AutoG&P:

Game BBStrong (G^{BBStrong})

```
1: $x, y \xleftarrow{\$} \mathbb{F}_p;\ z \leftarrow e(g_1, g_2);$ let $P = (g_1, g_2, g_2^x, g_2^y, z);$
2: $M[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q; S'[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q;$
   let $S[1..q] \leftarrow \{g_1^{s'_1}, \cdots, g_1^{s'_q}\};$
3: $R[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q;$
4: $m_1, \cdots, m_q \leftarrow \mathsf{A}_1(P);$
5: $(m^*, \sigma^*) \leftarrow \mathsf{A}_2(P, m_1, \cdots, m_q)$ with
6: Sign$(m_1, \cdots, m_q) = \{$
7: for $(i = 1; i \leq q; i++)$
8: $\{$
9: $r_i \xleftarrow{\$} \mathbb{F}_p; \sigma \leftarrow g_1^{\frac{1}{x+m_i+y*r_i}};$
10: guard$((x + m_i + y * r_i) \overset{?}{=} 0;\ \sigma_i \leftarrow 1);$
11: $\mathtt{write}(M, i, m_i); \mathtt{write}(S, i, \sigma_i);$
12: $\mathtt{write}(R, i, r_i);$
13: $\};$
14: $\};$
15: return $(m^* \neq \{m_1, \cdots, m_q\} \vee \sigma^* \neq \{\sigma_1, \cdots, \sigma_q\})$
```

Game G^{BBStrong}: In Line 1, the challenger picks the public/secret keypair. In lines 2,3,4 the challenger initializes the message and signature arrays, and the adversary picks query messages $m_1, \cdots, m_q$. In Line 5, the adversary is called and is asked to produce a valid message/signature pair. As before, adversaries maintain state between calls. The adversary is given access to a signing oracle, which returns valid signatures for the adversary's queries. The oracle has a guard to check if $(x + m_i + yr_i) = 0$, for some message m_i, where r_i denotes the internal random coins of the oracle, in which case the oracle returns $\sigma_i = 1$. In Line 15, the adversary returns a fresh message/signature pair. As a next step, we need to transform the G^{BBStrong} game such that the random variables used by the challenger are replaced in a way that is not obvious to the adversary. This is easy to do, since there are just three internal random variables used by the challenger, $x, y, r \in \mathbb{Z}_p^*$, using our rules, we can replace them by $x', y', r' \in \mathbb{Z}_p^*$, and the adversary cannot tell the difference. From that point on, it is easy to see that we can apply the q-SDH assumption, and match up the variables with the q-SDH simulator described below. We now describe the simulator built by AutoG&P, B, for the q-SDH assumption as applied to the strong signature scheme.

Simulator for q-SDH assumption in the strong scheme, B

```
1: $x, y \xleftarrow{\$} \mathbb{F}_p$; let $P = (\nu, \gamma, \gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_{q+1});$
2: $(m_1, \cdots, m_q) \leftarrow \mathsf{A}_1(P);$
3: $M[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q; S'[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q;$
   let $S[1..q] \leftarrow \{g_1^{s'_1}, \cdots, g_1^{s'_q}\};$
4: $R[1..q] \xleftarrow{\$} \mathsf{ArrFp}_q;$
5: $(m^*, \sigma^*) \leftarrow \mathsf{A}_2(P, m_1, \cdots, m_q)$ with
6: Sign $(m_1, \cdots, m_q) = \{$
7: for$(i = 1; i \leq q; i++)\{$
8: $r_i \xleftarrow{\$} \mathbb{F}_q;\ \sigma_i = \gamma^{\frac{1}{x+m_i+y*r_i}};$
9: guard$((x + m_i + y * r_i) \overset{?}{=} 0;\ \sigma_i = 1);$
10: $\mathtt{write}(M, i, m_i); \mathtt{write}(S, i, \sigma_i); \mathtt{write}(R, i, r_i);$
11: $\};$
12: $\};$
13: return $((\hat{e}(\sigma^*, \gamma_1 * \gamma_0^{m^*} * \gamma_{q+1}^r) \overset{?}{=} \nu)$
    $\mathtt{assert}\,((m^* \neq \{m_1, \cdots, m_q\}) \vee (\sigma^* \neq \{\sigma_1, \cdots, \sigma_q\})))$
```

Simulator B: In Line 1, the challenger chooses the public/secret keypair, $SK = (x, y); x, y \xleftarrow{\$} \mathbb{Z}_p^*$, and $PK = (\nu = e(g_1, g_2), \gamma = g_1, \gamma_0 = g_2, \gamma_1 = g_2^x, \gamma_2 = g_2^{x^2}, \cdots, \gamma_q = g_2^{x^q}), \gamma_{q+1} = g_2^y$. In Line 5, the adversary is called and is given access to a signing oracle. The oracle returns valid signatures on q messages of the adversary's choice. The oracle has a guard that checks for $(x + m_i + yr_i) = 0$, for some adversary-chosen m_i, in which case it returns $\sigma_i = 1$. Finally, in Line 13, we check if σ^* is a valid signature for an $m^* \notin \{m_1, \cdots, m_q\}$, or $\sigma^* \notin \{\sigma_1, \cdots, \sigma_q\}$.

VI. Boneh-Boyen-Shacham Group Signature Scheme

Boneh, Boyen and Shacham (BBS) introduced an efficient group signature scheme based on the q-SDH assumption and the decision linear assumption (DLIN) [24], in the random oracle model. Their technique basically involves a signer equipped with a solution to a q-SDH problem proving in zero knowledge their possession of the solution. In the zero knowledge proof, the prover's internal random coins and intermediate values get encrypted using a technique called *linear encryption*, that was shown to be CPA-secure under the DLIN assumption [24].

Group signature schemes have three security properties as defined by Bellare *et al.* [42]: correctness, full-anonymity, and full-traceability. The correctness property says that if the signer (prover) and verifier are both honest, then a valid group signature will always be accepted by the verifier. The anonymity property says that no party, except a group manager can identify which party created a signature. The traceability property says that no colluding subset of group members

TABLE III
EXPERIMENTAL RESULTS

Proof of Scheme	Lines of Code	Time (ms)
BB weak signature scheme	114	43
BB strong signature scheme	146	46
BBS group signature scheme-anon	69	25
BBS group signature scheme-traceability	68	32

should be able to create a signature that is un-openable and cannot be traced back to some member of the group. Bellare *et al.* had shown that full-anonymity and full-traceability imply all other requirements for group signatures (e.g., unforgeability, collusion-resistance, exculpability, framing, and more). Boneh *et al.* prove their construction possesses the properties of correctness, anonymity for signers, and traceability. In the BBS proofs, the adversary isn't allowed to query the Open function used to trace signatures. In other words, it is assumed the group manager is honest, and the adversary is not privy to the process of signatures being opened by the group manager, nor is he given the results of the opening process. The correctness property directly follows from the DLIN and q-SDH assumptions. In this paper, we automate both, the anonymity and traceability properties (see Appendix). Due to space constraints, we only give the initial game for each of these properties in Appendix; the detailed reductions and proofs are available in the full version of this paper; the source code is available on Github[2].

VII. EXPERIMENTS

We have implemented the proofs of the Boneh-Boyen weak and strong signature schemes, and the proof of the anonymity, traceability properties of the Boneh-Boyen-Shacham group signature scheme in AutoG&P, the results are tabulated in Table III. For all three schemes, the proof is discovered automatically. The BB weak signature scheme and BBS anonymity involve a reduction to the base assumptions. The strong scheme, and BBS traceability involve reductions to a weak adversary, and hence have comparable, fairly short running times. If we do a "long-form" reduction, i.e., not building on the security of the weak scheme, the running time for the strong scheme and traceability scheme would be around 30-40 milliseconds more. Our experiments were run on an Intel core $i3-7100$T (Dual Core, 3MB, 4T, 3.4GHz) running Ubuntu 16.04.

VIII. 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. Our motivation in choosing the Boneh-Boyen signature scheme was the fact that it is used as a building block in several high-level protocols notably attribute-based signature protocols [43]. Automating primitives such as signature schemes in the standard model would pave the way for automating proofs of families of several high-level protocols which use them. On another note, 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 EasyCrypt or Coq (only initial version had support) for verification.

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

REFERENCES

[1] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology - CRYPTO*, 2011, pp. 71–90.
[2] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *2006 IEEE Symposium on Security and Privacy (S&P)*, 2006, pp. 140–154.
[3] S. Halevi, "A plausible approach to computer-aided cryptographic proofs," Cryptology ePrint Archive, Report 2005/181, 2005.
[4] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.
[5] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *14th IEEE CSFW 2001*, 2001, pp. 82–96.
[6] K. Bhargavan, C. Fournet, A. D. Gordon, and N. Swamy, "Verified implementations of the information card federated identity-management protocol," in *Proceedings of ACM ASIACCS*, 2008, pp. 123–135.
[7] C. J. F. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification, 20th International Conference, CAV, Proceedings*, 2008, pp. 414–418.
[8] M. Aizatulin, A. D. Gordon, and J. Jürjens, "Extracting and verifying cryptographic models from C protocol code by symbolic execution," in *Proceedings of the 18th ACM CCS*, 2011, pp. 331–340.
[9] 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.
[10] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (the computational soundness of formal encryption)," *J. Cryptology*, vol. 15, no. 2, pp. 103–127, 2002.
[11] M. Aizatulin, A. D. Gordon, and J. Jürjens, "Computational verification of C protocol implementations by symbolic execution," in *Proceedings of the 19th ACM CCS*, 2012, pp. 712–723.
[12] G. Barthe, B. Grégoire, and S. Zanella-Béguelin, "Formal certification of code-based cryptographic proofs," in *36th ACM SIGPLAN-SIGACT POPL 2009*. ACM, 2009, pp. 90–101.
[13] G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, Y. Lakhnech, B. Schmidt, and S. Zanella-Béguelin, "Fully automated analysis of padding-based encryption in the computational model," in *Proceedings of the 2013 ACM CCS*, ser. CCS '13, 2013, pp. 1247–1260.
[14] I. labs, "The Coq proof assistant," https://coq.inria.fr/.
[15] A. Petcher and G. Morrisett, "The foundational cryptography framework," in *Principles of Security and Trust - 4th International Conference, POST, Proceedings*, 2015, pp. 53–72.
[16] G. Barthe, E. Fagerholm, D. Fiore, J. C. Mitchell, A. Scedrov, and B. Schmidt, "Automated analysis of cryptographic assumptions in generic group models," in *Advances in Cryptology - CRYPTO*, 2014, pp. 95–112.
[17] M. Ambrona, G. Barthe, and B. Schmidt, "Automated unbounded analysis of cryptographic constructions in the generic group model," in *Advances in Cryptology - EUROCRYPT 2016*, 2016, pp. 822–851.
[18] M. Ambrona, G. Barthe, R. Gay, and H. Wee, "Attribute-based encryption in the generic group model: Automated proofs and new constructions," in *Proceedings of the 2017 ACM CCS*, 2017, pp. 647–664.
[19] 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.
[20] D. Boneh and X. Boyen, "Efficient selective-id secure identity-based encryption without random oracles," in *Advances in Cryptology - EUROCRYPT*, 2004, pp. 223–238.
[21] B. Waters, "Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions," in *Advances in Cryptology - CRYPTO*, 2009, pp. 619–636.
[22] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology - CRYPTO*, 2001, pp. 213–229.
[23] D. Boneh and X. Boyen, "Short signatures without random oracles," in *Advances in Cryptology - EUROCRYPT*, 2004, pp. 56–73.
[24] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Advances in Cryptology - CRYPTO*, 2004, pp. 41–55.

[25] G. Barthe, D. Hedin, S. Z. Béguelin, B. Grégoire, and S. Heraud, "A machine-checked formalization of sigma-protocols," in *Proceedings of the 23rd IEEE CSF*, 2010, pp. 246–260.
[26] G. Barthe, B. Grégoire, Y. Lakhnech, and S. Z. Béguelin, "Beyond provable security verifiable IND-CCA security of OAEP," in *Proceedings of CT-RSA*, 2011, pp. 180–196.
[27] G. Barthe, F. Olmedo, and S. Z. Béguelin, "Verifiable security of boneh-franklin identity-based encryption," in *Provable Security - 5th International Conference, ProvSec 2011, Xi'an, China, October 16-18, 2011. Proceedings*, 2011, pp. 68–83.
[28] S. Z. Béguelin, G. Barthe, B. Grégoire, and F. Olmedo, "Formally certifying the security of digital signature schemes," in *30th IEEE Symposium on Security and Privacy (S&P)*, 2009, pp. 237–250.
[29] G. Barthe, M. Daubignard, B. M. Kapron, and Y. Lakhnech, "Computational indistinguishability logic," in *Proceedings of the 17th ACM CCS*, 2010, pp. 375–386.
[30] G. Barthe, E. Fagerholm, D. Fiore, A. Scedrov, B. Schmidt, and M. Tibouchi, "Strongly-optimal structure preserving signatures from type II pairings: Synthesis and lower bounds," in *Public-Key Cryptography - PKC*, 2015, pp. 355–376.
[31] 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*, ser. CCS '15, 2015, pp. 1370–1381.
[32] A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt, "A decision procedure for an extensional theory of arrays," in *16th Annual IEEE LICS, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, 2001, pp. 29–37.
[33] A. R. Bradley, Z. Manna, and H. B. Sipma, "What's decidable about arrays?" in *VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, 2006, pp. 427–442.
[34] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli, "Decision procedures for extensions of the theory of arrays," *Ann. Math. Artif. Intell.*, vol. 50, no. 3-4, pp. 231–254, 2007.
[35] R. E. Bryant, S. K. Lahiri, and S. A. Seshia, "Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions," in *CAV, Proceedings*, 2002, pp. 78–92.
[36] B. Dutertre, "Yices 2.2," in *(CAV'2014)*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559, 2014, pp. 737–744.
[37] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *TACAS 2008. Proceedings*, 2008, pp. 337–340.
[38] J. McCarthy, "Towards a mathematical science of computation," in *IFIP Congress*, 1962, pp. 21–28.
[39] S. Falke, C. Sinz, and F. Merz, "A theory of arrays with set and copy operations," in *10th International Workshop on Satisfiability Modulo Theories, SMT 2012, Manchester, UK, June 30 - July 1, 2012*, 2012, pp. 98–108.
[40] D. Jovanovic and C. Barrett, "Polite theories revisited," in *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17. Proceedings*, 2010, pp. 402–416.
[41] G. Nelson and D. C. Oppen, "Simplification by cooperating decision procedures," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 2, pp. 245–257, 1979.
[42] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," in *Advances in Cryptology - EUROCRYPT*, 2003, pp. 614–629.
[43] H. K. Maji, M. Prabhakaran, and M. Rosulek, "Attribute-based signatures," in *CT-RSA*, 2011, pp. 376–392.

Appendix

Definition A.1: (Bilinear Map): Let G_1 and G_2 be cyclic groups of prime order p, let g_1, g_2 be generators of G_1 and G_2 respectively. Let ψ be an efficiently computable isomorphism from G_2 to G_1 such that $g_1 = \psi(g_2)$, and let G_T be a third cyclic group such that $|G_1| = |G_2| = |G_T| = p$. Then $e : G_1 \times G_2 \to G_T$ is a bilinear map if it satisfies the following properties:

1) Bilinearity: For all $u \in G_1$, $v \in G_2$, and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$, and
2) Non-degeneracy: $e(g_1, g_2) \neq 1$.

Definition A.2: (q-Strong Diffie Hellman Assumption, q-SDH): The q-SDH assumption holds in (G_1, G_2) if, for a random $x \in \mathbb{Z}_p^*$, given the elements $(g_1, g_2, g_2^x, g_2^{x^2}, \cdots, g_2^{x^q})$, it is computationally infeasible to compute any pair of the form $(c, g_1^{\frac{1}{x+c}})$, with non-negligible probability, where $c \in \mathbb{Z}_p^*$.

Definition A.3: (Decision Linear Assumption, DLIN): The DLIN assumption holds in G_1 if, given the elements (u, v, h, u^a, v^b, h^c), for $u, v, h \in G_1$, and $a, b \in \mathbb{Z}_p$, it is computationally infeasible to decide if $c = a + b$, or c is random in $\mathbb{Z}_p$.

Definition A.4: (Linear Encryption Scheme, LE) [24]: A linear encryption scheme based on the DLIN assumption consists of the following three algorithms:

1) $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Pick an $x, y \in \mathbb{Z}_p$. Pick $u, v, h \in G_1$ such that $u^x = v^y = h$. Set $SK = (x, y), PK = (u, v, h)$.
2) $c \leftarrow \mathsf{Encrypt}(PK, m \in G_1)$: Pick $a, b \in \mathbb{Z}_p$. Compute $T_1 = u^a, T_2 = v^b, T_3 = m \cdot h^{a+b}$. Return $C = (T_1, T_2, T_3)$.
3) $m \leftarrow \mathsf{Decrypt}(c, SK)$: Compute $m \leftarrow T_3/(T_1^x \cdot T_2^y)$.

If the DLIN assumption holds the above linear encryption scheme is CPA-secure.

Theorem A.1: The conclusion of each rule in Definition 3.3 is satisfiable, iff one of its premises is satisfiable.

Proof: We consider each rule:

1) $\mathsf{Ext}, \mathsf{R\text{-}over\text{-}W}, \mathsf{Symm}$: Proof follows from the extensional array theory in Definition 3.1.
2) $\mathsf{W\text{-}Elim}, \mathsf{Partial\text{-}Eq}$: The proof follows from the definition of partial equations in Definition 3.2. We can see from that definition that if $a =_{\mathcal{I}} b$, and $p \notin \mathcal{I}$, then a and b agree on index p, and if $p \in \mathcal{I}$ means that a and b agree on every index, except p. For the other direction of the iff, if the premise has a model, so does the conclusion, as the conclusion is a subset of the premise.
3) Trans: If $a =_{\mathcal{I}} b$ and $a =_{\mathcal{I}'} c$ are true in some model, then from Definition 3.2, $b =_{\mathcal{I} \cup \mathcal{I}'}$ is also true in some model. If c agrees with b at every index except those in $\mathcal{I}$, then $p \notin \mathcal{I} \cup \mathcal{I}'$ implies that c agrees with a at p, and also that a agrees with b at p. Hence, c agrees with b at p. For the other direction of the iff, the conclusion is a subset of the premise, and hence has a model.
4) Subst: Evaluating an expression containing occurrences of array a gives the same result as evaluating the expression with the contents of a replaced with b at all indices, except those in set $\mathcal{I}$. The side-condition in the Subst rule ensures well-formedness.

■

AutoG&P provides algorithms to perform contextual reasoning and checking contextual equivalence of the form $\Gamma \models e =_\epsilon e'$, where Γ, ϵ are a set of axioms, and e and e' are a set of expressions. The set of axioms defined by Γ, ϵ are field axioms for finite fields of order p, $\mathbb{F}_p$, bilinear group axioms for groups G_i, and axioms for logical and bitwise operations. The algorithms for checking contextual equivalence between e and e' involves first normalizing e and e', and checking if e and e' are syntactically equal. Next, we compute the strongest postconditions at a given position, which is a collection of inequalities (random samplings and other commands) at every

TABLE IV
COMPUTING STRONGEST POSTCONDITION

Command	Effect on state
$c = x \xleftarrow{\$} t \setminus a$	$x \neq a$
$c = \mathsf{let}\, x = e$	$x = e$
$c = \mathsf{guard}(b)$	b
$c = \mathsf{assert}(ev)$	nquant(ev): inequalities not below a quantifier
$c = \mathsf{write}(a, i, \text{val})$	return $a[j] \leftarrow$ val

step leading up to the given position. To this end, a function, *conseq* was defined that characterized the effect of a command on the state of a (set of) variable(s). We extend the *conseq* function to deal with array write commands; the existing commands with our extension are given in Table IV. Once the strongest postcondition is computed, it is written in disjunctive normal form, and $\Gamma \models e =_\epsilon e'$ is checked for each disjunct separately. e and e' are then checked for syntactic equality, which involves normalizing variables of different data types (Boolean, bitstrings, $\mathbb{F}_p$, and $\mathbb{G}_i$) in e and e' in different ways. We deal with arrays of group elements by treating each indexed variable as an independent variable of type $\mathbb{G}_i$, and using AutoG&P's underlying normalization algorithm. In the future, we hope to optimize this step, and investigate if it might be faster to normalize the array as a whole, and what techniques would such a normalization entail.

We now give the BBS group signature scheme anonymity and traceability games. **Game GS-anon** (Ganon$^{\mathsf{GS},0}$)

```
1: $a_1, a_2, a_3, \gamma \xleftarrow{\$} \mathbb{F}_p$; let $P = (g_1, g_2, g_1^{a_1}, g_1^{a_2}, g_1^{a_3}, g_2^{\gamma})$;
2: $x \xleftarrow{\$} \mathbb{F}_p; A \leftarrow g_1^{1/(\gamma+x)}$; let $K_0 = (A, x)$;
3: $b \xleftarrow{\$} \mathbb{B}; x' \xleftarrow{\$} \mathbb{F}_p; A' \leftarrow g_1^{x'}; K_1 = (A', x'); q_h \xleftarrow{\$} \mathbb{F}_p$;
4: $m \leftarrow \mathsf{A}_1()$ with
5: $H[1..q_h] \xleftarrow{\$} \mathsf{ArrFp}_q$;
6: for $(i = 1; i \leq q_h; i++)$
7: {
8: let $r \xleftarrow{\$} \mathbb{F}_p$;
9: $\mathtt{write}(H, i, H(r))$;
10: };
11: $b' \leftarrow \mathsf{A}_2(P, m, H[1..q_h], K_0, K_1, (b?K_0 : K_1))$ with
12: Sign$(m) = \{$
13: $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p$;
14: let $T_1 = g_1^{(a_1 * \alpha)}$; let $T_2 = g_1^{(a_2 * \beta)}$;
15: let $T_3 = A * g_1^{(a_3 * (\alpha+\beta))}$;
16: let $\delta_1 = x * \alpha$; let $\delta_2 = x * \beta$;
17: $r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \xleftarrow{\$} \mathbb{F}_p$;
18: let $R_1 = g_1^{(a_1 * r_\alpha)}$; let $R_2 = g_1^{(a_2 * r_\beta)}$;
19: let $R_4 = T_1^{r_x} * g_1^{(-a_1 * r_{\delta_1})}$;
20: let $R_5 = T_2^{r_x} * g_1^{(-a_2 * r_{\delta_2})}$;
21: let $R_3 = \hat{e}(T_3, g_2)^{r_x} * \hat{e}(g_1^{a_3}, g_2^{\gamma})^{(-r_\alpha - r_\beta)}$
        $* \hat{e}(g_1^{a_3}, g_2)^{(-r_{\delta_1} - r_{\delta_2})}$;
22: let $c = H(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$;
23: let $s_\alpha = r_\alpha + c * \alpha$; let $s_\beta = r_\beta + c * \beta$;
24: let $s_x = r_x + c * x$;
25: let $s_{\delta_1} = r_{\delta_1} + c * \delta_1$; let $s_{\delta_2} = r_{\delta_2} + c * \delta_2$;
26: let $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$;
27: return $\sigma$
28: };
```

In Line 4, A_1 queries a random oracle (modeled as hash function H); q_h is the bound on hash function queries. The challenger responds with elements selected randomly from $\mathbb{Z}_p$ (in the code, we maintain an additional buffer to check for and ensure the oracle responds consistently to identical queries, but have omitted that part here). These values are then passed on to the adversary A_2. In Line 11, the adversary gets the public key, the message, and the responses to its hash function queries. Additionally, if $b = 0$, the challenger gives K_0 as input to the adversary, and K_1 is given to adversary if $b = 1$ (A_2, of course, does not know b).

The traceability game basically reduces the hardness of the adversary forging a signature that traces back to a member of the group, without knowing that group member's signing key to the hardness of the G^{BB} game. The proof search proceeds similar to the $GS-$anon game, except that the reduction is to a G^{BB} adversary. We do not give all the steps and the reductions of the anonymity and traceability games, due to space constraints (available in the full version).

Game GS-trace ($G-$trace$^{\mathsf{GS},0}$)

```
1: $a_1, a_2, a_3, \xi_1, \xi_2, \gamma \xleftarrow{\$} \mathbb{F}_p$; let $P = (g_1, g_2, g_1^{a_1}, g_1^{a_2}, g_1^{a_3}, g_2^{\gamma})$;
2: $X[1..q-1] \xleftarrow{\$} \mathsf{ArrFp}_q$;
     let $A[1..q-1] \leftarrow [g_1^{1/(\gamma+X[1])}, \ldots, g_1^{1/(\gamma+X[q-1])}]$;
3: $m \leftarrow \mathsf{A}_1()$ with
4: $H[1..q_h] \xleftarrow{\$} \mathsf{ArrFp}_q$;
5: for $(i = 1; i \leq q_h; i++)$
6: {
7: let $r \xleftarrow{\$} \mathbb{F}_p$;
8: $\mathtt{write}(H, i, H(r))$;
9: };
10: $\sigma \leftarrow \mathsf{A}_2(P, m, \xi_1, \xi_2, (A[i^*], X[i^*]))$ with
11: Sign$() = \{$
12: $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p$;
13: let $T_1 = g_1^{(a_1 * \alpha)}$; let $T_2 = g_1^{(a_2 * \beta)}$;
14: let $T_3 = A[i^*] * g_1^{(a_3 * (\alpha+\beta))}$;
15: let $\delta_1 = X[i^*] * \alpha$; let $\delta_2 = X[i^*] * \beta$;
16: $r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \xleftarrow{\$} \mathbb{F}_p$;
17: let $R_1 = g_1^{(a_1 * r_\alpha)}$; let $R_2 = g_1^{(a_2 * r_\beta)}$;
18: let $R_4 = T_1^{r_x} * g_1^{(-a_1 * r_{\delta_1})}$;
19: let $R_5 = T_2^{r_x} * g_1^{(-a_2 * r_{\delta_2})}$;
20: let $R_3 = \hat{e}(T_3, g_2)^{r_x} * \hat{e}(g_1^{a_3}, g_2^{\gamma})^{(-r_\alpha - r_\beta)}$
        $* \hat{e}(g_1^{a_3}, g_2)^{(-r_{\delta_1} - r_{\delta_2})}$;
21: let $c = H(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$;
22: let $s_\alpha = r_\alpha + c * \alpha$; let $s_\beta = r_\beta + c * \beta$;
23: let $s_x = r_x + c * X[i^*]$;
24: let $s_{\delta_1} = r_{\delta_1} + c * \delta_1$; let $s_{\delta_2} = r_{\delta_2} + c * \delta_2$;
25: let $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$;
26: return $\sigma$
27: };
28: let $A[i^*] = T_3/(T_1^{\xi_1}, T_2^{\xi_2})$;
```

```
return $(A[i^*], X[i^*])$;
};
```