Are These Pairing Elements Correct? Automated Verification and Applications

Susan Hohenberger* Satyanarayana Vusirikala†

December 2, 2019

Abstract

Using a set of pairing product equations (PPEs) to verify the correctness of an untrusted set of pairing elements with respect to another set of trusted elements has numerous cryptographic applications. These include the design of basic and structure-preserving signature schemes, building oblivious transfer schemes from "blind" IBE, finding new verifiable random functions and keeping the IBE/ABE authority "accountable" to the user.

A natural question to ask is: are all trusted-untrusted pairing element groups in the literature PPE testable? We provide original observations demonstrating that the answer is no, and moreover, it can be non-trivial to determine whether or not there exists a set of PPEs that can verify some pairing elements with respect to others. Many IBE schemes have PPE-testable private keys (with respect to the public parameters), while others, such as those based on dual-system encryption, provably do not.

To aid those wishing to use PPE-based element verification in their cryptosystems, we devised rules to systematically search for a set of PPEs that can verify untrusted elements with respect to a set of trusted elements. We prove the correctness of each rule and combine them into a main searching algorithm for which we also prove correctness. We implemented this algorithm in a new software tool, called AutoPPE. Tested on over two dozen case studies, AutoPPE found a set of PPEs (on schemes where they exist) usually in just a matter of seconds. This work represents an important step towards the larger goal of improving the speed and accuracy of pairing-based cryptographic design via computer automation.

1 Introduction

Computer automation is showing great potential to improve the speed and accuracy of the cryptographic design process. Over the past several years, a host of new software tools, e.g., [7, 6, 5, 15, 18, 16, 12, 13, 11, 17], were made public for handling a variety of cryptographic tasks, including design, proof generation, and proof verification. Automation is particularly compelling for these tasks, which are often both complex and tedious, and where a single error can compromise the entire system.

Many of these tools focus on the pairing-based algebraic setting, since it is popular both for its efficiency and functionality. In this work, we focus on automating a *novel* cryptographic design task in this setting, which we call *pairing-product equation (PPE) testability.* Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p. Recall that a *pairing* is an efficient map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, such that for all $g \in G_1$, $h \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, h^b) = e(g, h)^{ab}$. Following [33], a *pairing product equation* (PPE) over variables $Z, \{X_i\}_{i=1}^m, \{Y_i\}_{i=1}^n$ is an equation of the form

$$Z \cdot \prod_{i=1}^{n} e(A_i, Y_i) \cdot \prod_{i=1}^{m} e(X_i, B_i) \cdot \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{\gamma_{ij}} = 1,$$

*Johns Hopkins University. Email: susan@cs.jhu.edu. Supported by Office of Naval Research contract N00014-19-1-2294, NSF Frontier CNS-1414023, NSF Medium CNS-1908181, a Microsoft Faculty Fellowship and a Packard Foundation subaward via UT Austin.

†University of Texas at Austin. Email: satya@cs.utexas.edu. Supported by a UT Austin Provost Fellowship, NSF grants CNS-1908611 and CNS-1414082, DARPA SafeWare and the Packard Foundation.

where $A_i, X_i \in \mathbb{G}_1, B_i, Y_i \in \mathbb{G}_2, Z \in \mathbb{G}_T, \gamma_{ij} \in \mathbb{Z}_p$.

Informally, PPE testability captures the commonplace task of figuring out a method of verifying one set of group elements with respect to another set using the pairing map. This is extremely useful when designing new pairing-based constructions. For instance, the need to verify a signature with respect to a public key and message; or to verify a verifiable random function output and proof with respect to a public key. Let's see more examples after formalizing this concept more crisply.

In a nutshell, our research discovered examples illustrating that deciding whether a given cryptographic scheme supports PPE testability is highly non-trivial. There are natural examples where the answer is yes, provably no and even unknown. Our contributions in this work are: (1) formalizing the concept of PPE testability, (2) developing novel techniques to search for a PPE-based verification procedure (which we will call a *PPE testing set*), (3) proving the correctness of this searching algorithm, (4) implementing this algorithm as an open source software tool called *AutoPPE*, (5) reporting on the performance and accuracy of AutoPPE on over two dozen case studies and (6) documenting provably non-PPE-testable instances.

1.1 Deciding PPE Testability is Non-Trivial

Let's explore our objective in more detail. Let a *PPE problem instance* be a set of pairing parameters, a set of multivariate polynomials $\mathbf{f} = (f_1, \ldots, f_m)$ over variables $u = (u_1, \ldots, u_n)$ in $\mathbb{Z}_p$, a sequence of pairing group identifiers $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$, a set $\mathsf{Fixed} \subseteq [1, n]$ and a set $\mathsf{Trusted} \in [1, m]$. This instance corresponds to a set of group elements of the form $\mathbf{F} = (g_{\alpha_1}^{f_1(\mathbf{u})}, \ldots, g_{\alpha_m}^{f_m(\mathbf{u})})$, where the variables $u_i \in \mathbf{u}$ for $i \in \mathsf{Fixed}$ are chosen by the "trusted" source and the polynomials $f_i \in \mathbf{F}$ for $i \in \mathsf{Trusted}$ are only over these fixed variables.

The goal of our work is that given a PPE problem instance, is there an efficient algorithm for deciding the existence of (and if yes, producing) a set of PPEs that will verify that $F_j = g_{\alpha_j}^{f_j(\mathbf{u})}$, for all $j \notin \mathsf{Trusted}$, for any setting of $\mathbf{u}$. Intuitively, if we assume the trusted elements are correctly formed, can we verify that the untrusted ones are correctly formed too, using PPEs? (We will not concern ourselves with the case that the *trusted* elements are not well formed.) If the answer is yes, then we say that this PPE problem instance is *PPE testable*. In Section 2, we will provide formal definitions for a PPE problem instance and PPE testability.

Let's explore these notions with some examples set in the Type I pairing setting, where $\mathbb{G}_1 = \mathbb{G}_2$. Suppose we have public parameters (g, g^a, g^b) and want to verify if a value T is g^{ab} or not. Can we do this with a PPE? Easy: check that $e(g^a, g^b) = e(T, g)$. Here, we'd have $\mathbf{f} = (f_1 = 1, f_2 = u_1, f_3 = u_2, f_4 = u_1 u_2)$ with $\boldsymbol{\alpha} = (\mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1)$, $\mathbf{u} = (u_1, u_2)$, $\mathsf{Fixed} = \{1, 2\}$ and $\mathsf{Trusted} = \{1, 2, 3\}$. Here's another PPE testable example with some variables not in Fixed. Suppose the public parameters are (g, g^a) and we want to test the elements $(T_1, T_2) = (g^r, g^{ar})$. Here, we'd have $\mathbf{f} = (f_1 = 1, f_2 = u_1, f_3 = u_2, f_4 = u_1 u_2)$ with $\boldsymbol{\alpha} = (\mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1)$, $\mathbf{u} = (u_1, u_2)$, $\mathsf{Fixed} = \{1\}$ and $\mathsf{Trusted} = \{1, 2\}$ and the PPE as $e(g^a, T_1) = e(T_2, g)$.

Next, let's see a simple example that is *not* PPE testable. Suppose we have public parameters (g, g^a, g^b, g^c) and want to verify if a value T is g^{abc} or not. Here, we'd have $\mathbf{f} = (f_1 = 1, f_2 = u_1, f_3 = u_2, f_4 = u_3, f_5 = u_1 u_2 u_3)$ with $\boldsymbol{\alpha} = (\mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1, \mathbb{G}_1)$, $\mathbf{u} = (u_1, u_2, u_3)$, $\mathsf{Fixed} = \{1, 2, 3\}$ and $\mathsf{Trusted} = \{1, 2, 3, 4\}$. However, this problem is the Decisional Bilinear Diffie-Hellman (DBDH) problem, so this would not be a PPE testable instance in any group where the DBDH assumption holds.

There are many encryption systems where an authority distributes private keys to users, and the user would like to verify that their key was correctly formed (i.e., per the key generation procedure). This comes up in IBE to signature design [22], realizing "blind IBE" to build oblivious transfer [32], and keeping the IBE authority "accountable" to the user [30, 31]. We discuss these applications in more detail soon, but think a moment about how the reader would approach PPE testing for the Waters dual system IBE [46]. Set in a Type I group, the Setup algorithm chooses random generators g, v, v_1, v_2, w, u, h and exponents $a_1, a_2, b, \alpha \in \mathbb{Z}_p$, sets $\tau_1 = v v_1^{a_1}$ and $\tau_2 = v v_2^{a_2}$, and then publishes the public parameters as $\mathsf{pp} = (g^b, g^{a_1}, g^{a_2}, g^{ba_1}, g^{ba_2}, \tau_1, \tau_2, \tau_1^b, \tau_2^b, w, u, h, e(g, g)^{\alpha a_1 b})$. How would one then verify a private key for

identity id of the form $(D_1, \ldots, D_7, K, t)$, where $r_1, r_2, z_1, z_2, t \in \mathbb{Z}_p$, $r = r_1 + r_2$ and

$$\begin{array}{rrr} D_1 = g^{\alpha a_1} v^r & D_2 = g^{-\alpha} v_1^r g^{z_1} & D_3 = (g^b)^{-z_1} \\ D_4 = v_2 g^{z_2} & D_5 = (g^b)^{-z_2} & D_6 = g^{r_2 b} \\ D_7 = g^{r_1} & K = (u^{\mathsf{id}} w^t h)^{r_1} & \end{array}$$

This is just the IBE scheme and not the HIBE! And lest the reader shrug this off as too complicated to actually be of interest for PPE testability, we counter that there are documented examples, e.g., Abe et al. [1], verifying derivatives of these private keys with PPEs to use as the base of a structure-preserving signature scheme. Indeed, when looking at IBEs of this complexity and even more advanced attribute-based encryption schemes, etc., we hope to persuade the reader to appreciate the value of having a software tool do this work rather than a human.

So, ultimately, why did Abe et al. [1] settle for a (less secure) derivative scheme instead of devising PPEs to test the Waters09 IBE private keys? To our surprise, we have the following:

Claim 1.1 (Informal)**.** *The public parameters and private keys for the Waters09 IBE [46] are not PPE testable, under the DBDH and Decision Linear assumptions.*

Waters proves this IBE system secure under the DBDH and Decision Linear assumptions. However, as part of his *dual system* security proof, he argues that under these assumptions no polynomial-time adversary can distinguish a real private key (generated by the key generation algorithm) from a "semi-functional" private key (used in the proof of security). In his construction, there is no overlap between the real and semi-functional key spaces. Thus, to be PPE testable, there must exist a PPE testing set that accepts all real private keys, but rejects all semi-functional keys. However, Waters argues that, under DBDH and Decision Linear, there is no efficient algorithm capable of making this bifurcation.

This is certainly a curious counterexample to the thinking that this problem would be easy, and it was not the only curious example we discovered (see the discussion of the Boyen-Waters anonymous IBE and the Dodis VRF in Section 5.6). However, we were encouraged by our results that show that the vast majority of our test cases were PPE testable and, moreover, that our searching algorithm was able to find them in usually a matter of seconds. We describe how we systemically search for a PPE testing set in Section 4.

1.2 Applications of PPE Testability

Pairing-based schemes are prevalent for their efficiency and functionality. There are a host of applications where one wants to verify some Untrusted pairing elements with respect to a set of Trusted elements. For starters, this is the basic goal of a signature scheme where the purported signature is Untrusted and the verification key and message are Trusted. Likewise, in a verifiable random function, one is given a function output with a proof that are Untrusted and one needs to verify them with respect to a Trusted public key. One can see expanding this to verifying anonymous credentials, e-cash and more.

There are also several interesting applications for this when using identity-based encryption (IBE) schemes. First, per Naor's observation in [22], any identity-based encryption (IBE) scheme gives rise to a signature scheme, where the verification key is the public parameters and the signature on message m is a private key for identity m. Naor's suggested verification procedure is to encrypt a random message under identity m and then try to decrypt it using the purported signature as the private key. This randomized and rather inefficient verification procedure is often replaced in practice by a direct verification of the signature elements using a set of pairing-product equations. The signature schemes derived from IBEs with PPE-verification often possess a "structure-preserving" feature that make them particularly useful and efficient as a building block in larger systems, such as anonymous credentials.

A second example is the *Blind IBE* used to build adaptive oblivious transfer schemes by Green and Hohenberger [32]. In their oblivious transfer scheme, the Sender acts as the master authority in an IBE scheme and encrypts each message m_i under identity i. To retrieve a message m_j, the Receiver engages in a "blind" key extraction protocol with the Sender, so that at the end of the protocol the Receiver obtains the

private key for identity j and the Sender does not learn j. As part of this blind key extraction protocol, the Receiver uses a set of PPEs to verify the correctness of the private key for identity j.

A third example is *Accountable Authority IBE* introduced by Goyal [30] and expanded on by Goyal, Lu, Sahai and Waters [31], where should a decryption key or program for that user's identity be found online, there exists a mechanism for the user to prove to a judge that it was the authority and not her that leaked this information. Again, since the user does not fully trust the authority that provides her with a private key, the authors require an efficient method (via a set of PPEs) to verify the well-formedness of the key she obtains.

Finally, this goal of having the user verify the private key given by a master authority translates over nicely as well to the ciphertext-policy attribute-based encryption setting, where the authority purports to give the user a key representing a set of attributes. As the complexity of the system increases (to ABE and beyond), the ability to derive the set of PPEs *automatically* becomes increasingly attractive.

1.3 Related Work

The effort to automate cryptographic design and verification tasks has been gaining momentum and enjoying much success in the last few years.

In 2014, Barthe, Fagerholm, et al. [13] put forward the GGA tool for automatically analyzing (bounded) cryptographic assumptions in the generic group model. This tool was extended to unbounded assumptions by Ambrona, Barthe and Schmidt [10]. Shortly thereafter, Ambrona, Barthe, Gay and Wee [9] showed how to apply this computer-aided reasoning to the design of complex cryptographic constructions, such as attribute-based encryption systems. We use the GGA tool as one piece of the AutoPPE tool.

In other related work, Barthe, Fagerholm, Fiore, Scedrov, Schmidt and Tibouchi [14] built an automated tool to design optimal structure-preserving signatures in Type II pairing groups. As they state [14], their "tool can generate automatically and exhaustively all valid structure-preserving signatures within a user-specified search space, and analyze their (bounded) security in the generic group model." Interestingly, some of the logic they employ in their synthesis algorithm closely resembles our Rule 1 presented in Section 4.2.1.

The AutoPPE tool is designed to be interoperable with several existing open source automation tools, built by a community of authors, such as: AutoBatch [7, 8] (for batching the verification of PPEs), AutoStrong [6] (for compiling a signature scheme secure under the standard definition into one that is strongly secure), AutoGroup+ [6, 5] (for translating a Type-I pairing scheme into a Type-III pairing scheme; we also note some nice work on alternative methods for this translation including IPConv [2, 3], although these are not available as open source at this time), AutoG&P [17] (for automatically proving security of cryptographic constructions based on pairing-based assumptions), and AutoLWE [15] (for semi-automatically proving security of cryptographic constructions based on the Learning with Errors assumption).

Most of the above examples are for the public key setting, although there have also been elegant automation results for blockciphers [41] and authenticated encryption [35] as well.

There is also a large-body of impressive work on machine-based cryptographic proof verification, such as Cryptoverif [18], CertiCrypt [16], EasyCrypt [12] and other tools, e.g. [11]. Tying these two bodies of work together, Barthe et al. [17] provided a tool that translates the proofs output by AutoG&P into a format verifiable by EasyCrypt and similarly Akinyele et al. [4] showed that the proofs output by AutoBatch can be automatically verified by EasyCrypt.

2 Preliminaries

We define the algebraic setting and notation used in throughout this work.

2.1 Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p[1]. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an admissible *pairing* (also called a *bilinear map*) if it satisfies the following three properties:

1. Bilinearity: for all $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, it holds that $e(g_1^a, g_2^b) = e(g_1^b, g_2^a) = e(g_1, g_2)^{ab}$.
2. Non-degeneracy: if g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, resp., then $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.
3. Efficiency: there exists an efficient method that given any $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, computes $e(g_1, g_2)$.

A pairing generator PGen is an algorithm that on input a security parameter 1^λ, outputs the parameters for a pairing group $(p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order $p \in \Theta(2^\lambda)$ where g_1 generates $\mathbb{G}_1$, g_2 generates $\mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an admissible pairing. The above pairing is called an *asymmetric* or Type-III pairing. In Type-II pairings, there exists an efficient isomorphism ψ from $\mathbb{G}_1$ to $\mathbb{G}_2$ or such an isomorphism ϕ from $\mathbb{G}_2$ to $\mathbb{G}_1$ but not both. In *symmetric* or Type-I pairings, efficient isomorphisms ψ and ϕ both exist, and thus we can consider it as though $\mathbb{G}_1 = \mathbb{G}_2$. In this work, we support any of these types of pairings. We will typically refer to Type III pairings in our text, since they are general and typically the most efficient choice for implementation, but our software tool in Section 5 can handle any type.

Given pairing parameters $(p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, we follow prior definitions [33] to define a *pairing product equation* over variables $Z, \{X_i\}_{i=1}^m, \{Y_i\}_{i=1}^n$ as an equation of the form

$$Z \cdot \prod_{i=1}^{n} e(A_i, Y_i) \cdot \prod_{i=1}^{m} e(X_i, B_i) \cdot \prod_{i=1}^{m} \prod_{j=1}^{n} e(X_i, Y_j)^{\gamma_{ij}} = 1,$$

where $A_i, X_i \in \mathbb{G}_1, B_i, Y_i \in \mathbb{G}_2, Z \in \mathbb{G}_T, \gamma_{ij} \in \mathbb{Z}_p$.

We sometimes rearrange the terms of a PPE to improve readability. The identity element for all groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ (which we here treat as multiplicative groups) will be defined as 1.

2.2 Notation

We let $[1, n]$ be shorthand for the set $\{1, \ldots, n\}$. We use $\mathbf{v}$ to denote a vector and $\mathbf{v}_i$ to denote the i-th element. For a vector $\mathbf{v}$ of length n and a subset $U \subseteq [1, n]$, we denote $\mathbf{v}^U$ as the set of elements v_i for $i = 1, \ldots, n$ where $i \in U$. Similarly $\mathbf{v}^{\overline{U}}$ denotes the subset of elements $\mathbf{v}_i$ for $i = 1, \ldots, n$ where $i \notin U$. Let us denote the set of pairing group identifiers $\{1, 2, T\}$ by $\mathcal{I}$. Let x, y be polynomials over variables in $(u_1, \ldots, u_n)$, then by $x \equiv y$, we mean that x and y are equivalent polynomials.

3 Defining PPE Testing Concepts

Let us now formalize our focus problem from the introduction.

Definition 3.1 (PPE Problem Instance)**.** *A* pairing product equation (PPE) problem instance Π *consists of*

- *pairing parameters* $\mathcal{G} = (p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$,
- *positive integers* n, m,
- *multivariate rational polynomials* $\mathbf{f} = (f_1, \ldots, f_m)$ *over* n *variables in* $\mathbb{Z}_p$ *denoted* $\mathbf{u} = (u_1, \ldots, u_n)$,
- *a sequence of pairing group identifiers in* $\mathcal{I} = \{1, 2, T\}$ *denoted* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$,
- *a set* $\mathsf{Fixed} \subseteq [1, n]$ *and*
- *a set* $\mathsf{Trusted} \subseteq [1, m]$.

with the restriction that if $i \in \mathsf{Trusted}$*, then* f_i *is a multivariate rational polynomial over the set of variables in* $\mathbf{u}^{\mathsf{Fixed}}$.

[1] In this work, we restrict ourselves to prime-order pairings. For discussion on other settings, refer to Section 5.6

The pairing parameters above can also indicate the type of pairing group (e.g., I, II or III). We remark that one can intuitively view the elements indicated by the Trusted set as a set of trusted (e.g., public) parameters and the set of elements not in Trusted as some elements one wants to verify with respect to the Trusted set (e.g., an IBE/ABE private key). The polynomials representing these elements are comprised of a set of variables; those variables in Fixed can be thought of as being chosen at public parameter setup time, while those variables not in Fixed correspond to values chosen later (e.g., during private IBE/ABE key generation). We sometimes denote the set of polynomials not in Trusted by untrusted set, and the set of variables not in Fixed by unfixed variables.

Definition 3.2 (PPE Challenge)**.** *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ *be a PPE problem instance as in Definition 3.1. Let* $\mathbf{F} = (F_1, \ldots, F_m)$ *be comprised of pairing group elements, where each* F_i *is in group* $\mathbb{G}_{\alpha_i}$*. We say that* $\mathbf{F}$ *is a* challenge *to PPE instance* Π*. We define classifications for this challenge as follows:*

- $\mathbf{F} = (F_1, \ldots, F_m)$ *is a* YES *challenge if there exists an assignment to variables* $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}_p^n$ *such that for all* i*,* $F_i = g_{\alpha_i}^{f_i(\mathbf{u})}$*.*
- $\mathbf{F} = (F_1, \ldots, F_m)$ *is a* NO *challenge if it is not a* YES *challenge* and *there exists an assignment to* $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}_p^n$ *such that for all* $i \in \mathsf{Trusted}$*,* $F_i = g_{\alpha_i}^{f_i(\mathbf{u})}$*.*
- $\mathbf{F} = (F_1, \ldots, F_m)$ *is an* INVALID *challenge if it is neither a* YES *nor* NO *challenge.*

We view a YES challenge as being a valid trusted/untrusted (e.g., public key/private key) pair, i.e., one that could have come from the distribution dictated by the instance parameters. We view a NO challenge as having trusted information according to the instance distribution, but where the untrusted elements to be verified do not fall into their proper distribution space. In an INVALID challenge, the supposedly trusted elements are not drawn from the proper distribution (e.g., the public parameters are not correct), and therefore, we make no attempt to verify with this challenge.

Definition 3.3 (PPE Testable and Testing Set)**.** *A PPE problem instance* Π *is said to be* PPE testable *if and only if there exists a set of pairing product equations* T *such that each equation in* T *is simultaneously satisfied for all* YES *challenges and for all* NO *challenges, at least one equation in* T *is not satisfied. (There are no conditions on the behavior for* INVALID *challenges.) For any PPE problem instance* Π*, we call such a set of pairing product equations* T *a* testing set*. A testing set for a PPE problem instance need not be unique.*

Our subsequent goal will be to search for a testing set for a given PPE problem instance. In Section 1, we discussed how some natural constructions of cryptosystems exhibit pubic parameter and private key pairs that are PPE testable problem instances, whereas other natural examples (e.g., encryption systems based on dual system techniques) are provably *not* PPE testable.

4 Searching for a PPE Testing Set

Recall from the introduction our high-level algorithm to search for a testing set Q of a PPE problem. The input is a PPE problem Π and there are two possible types of outputs. Either it will output that Π is PPE testable and, to confirm this, it will produce one testing set Q or it will output the special response unknown. In the latter case, no determination about whether Π is PPE testable or not can be concluded. This algorithm has one-sided correctness, where the guarantee for this algorithm is that if it outputs that Π has testing set Q, this will be true.

The algorithm proceeds in a sequence of steps, where in each step it (attempts to) "reduce the complexity" of its input, by adding a polynomial f_i to the set Trusted and possibly adding a variable u_i to the set Fixed. We establish rules for when an item can be moved into one of these sets, how this movement contributes to the search for Q and argue that these rules preserve the PPE testability of the input problem. At the end, if we can obtain $\mathsf{Trusted} = [1, m]$, then we will have found a testing set. At any time before the end, if none of the movement rules can be applied, the algorithm aborts and outputs unknown.

4.1 Review on Computing Completion Lists for a List of Polynomials

Our rules will make use of completion lists in the pairing setting as described by Barthe et al. [13]. Consider any list $\mathbf{f} = [f_1, \ldots, f_k]$ of polynomials along with a sequence of identifiers $\alpha_1, \cdots, \alpha_k$, where $\alpha_i \in \mathcal{I} = \{1, 2, T\}$ for all $i \leq k$. For any $j \in \mathcal{I}$, let $\mathbf{t}_i = \{f_j : \alpha_j = i\}$. We now recall the notion of completion $\mathsf{CL}(\mathbf{f}) = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\}$ of the list $\mathbf{f}$ of polynomials with respect to a group setting [13]. Intuitively, $\mathsf{CL}(\mathbf{f})$ is the list of all polynomials that can be computed by an adversary by applying pairing and isomorphism operations, when he has access to the elements in group $\mathbb{G}_i$ corresponding to the polynomials in $\mathbf{t}_i$ for $i \in I$.

Reception List

Input: Pairing info $\mathcal{G}$, Lengths $|\mathbf{t}_1|, |\mathbf{t}_2|, |\mathbf{t}_T|$
Output: Reception lists $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_T$

1. for each $i \in \{1, 2, T\}$, initialize $\mathbf{r}_i$ with $|\mathbf{t}_i|$ number of fresh variables, i.e., let $\mathbf{r}_i = \{w_{i,1}, \cdots, w_{i,|\mathbf{t}_i|}\}$
2. If an isomorphism $\psi : G_1 \to G_2$ exists, then $\mathbf{r}_2 := \mathbf{r}_2 \cup \mathbf{r}_1$
3. If an isomorphism $\phi : G_2 \to G_1$ exists, then $\mathbf{r}_1 := \mathbf{r}_1 \cup \mathbf{r}_2$
4. $\mathbf{r}_T := \mathbf{r}_T \cup \{h_1 h_2 : h_1 \in \mathbf{r}_1, h_2 \in \mathbf{r}_2\}$

Figure 1: Algorithm to find reception list of a list of polynomials

We now describe an algorithm to compute the completion $\mathsf{CL}(\mathbf{f})$, which is taken from [13] and handles pairing groups. The algorithm proceeds in two steps. In the first step, it computes the reception lists $\{\mathbf{r}_i\}_{i \in \mathcal{I}}$. The elements of the reception lists are monomials over variables $w_{i,j}$ for $i \in \mathcal{I}$, $j \in |\mathbf{t}_i|$ and are computed as shown in Figure 1.

The monomials characterize which products of elements in $\mathbf{t}$ the adversary can compute by applying pairing operations. The result of the first step is independent of the elements in the lists $\mathbf{t}$ and only depends on the lengths of the lists. In the second step, it computes the actual polynomials from the reception lists as

$$\mathbf{s}_i = [m_1(\mathbf{t}), \ldots, m_{|\mathbf{r}_i|}(\mathbf{t})] \text{ for } [m_1, \ldots, m_{|\mathbf{r}_i|}] = \mathbf{r}_i,$$

where every m_k is a monomial over the variables $w_{i,j}$ and $m_k(\mathbf{t})$ denotes the result of evaluating the monomial m_k by substituting $w_{i,j}$ with $\mathbf{t}_i[j]$ for $i \in \mathcal{I}$ and $j \in |\mathbf{t}_i|$.

4.2 Rules for "Reducing" the Complexity of a PPE Problem Instance

We now describe two rules for reducing the complexity of a PPE instance, whereby we mean reducing the number of items left to verify, i.e., corresponding to the elements represented by polynomials not in the set $\mathsf{Trusted}$.

4.2.1 Rule 1: Move element to Trusted with all Fixed variables

Rule 1 is described in Figure 2. We note that similar logic to this was previously employed in an automated tool focused on synthesizing optimal structure-preserving signature schemes in Type II pairing groups by Barthe, Fagerholm, Fiore, Scedrov, Schmidt and Tibouchi [14]. We generalize for our related, but different goal. Given a PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ and an index $k \in [1, m]$, Rule 1 can possibly be applied if $k \notin \mathsf{Trusted}$ and the polynomial $f_k \in \mathbf{f}$ consists only of variables $u_i \in \mathbf{u}$ where $i \in \mathsf{Fixed}$ (these conditions are necessary, but not sufficient).

Let $(C, \Pi') = \mathsf{Rule1}(\Pi, k)$ for an input on which the rule is successfully applied. Let the set of all possible testing sets of PPE problems Π and Π' be denoted Q_Π and $Q_{\Pi'}$ respectively. We now prove that a testing set for Π can be derived from any testing set in $Q_{\Pi'}$; we call this the correctness of Rule 1. Informally, Rule 1 will not flip a PPE problem from non-testable to testable.

Description of Rule 1

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ and an integer $k \in [1, m]$.
Output: A PPE C and a PPE problem Π', or the symbol $\perp$ (meaning could not apply rule).
Steps of $\mathsf{Rule1}(\Pi, k)$:

1. If $k \in \mathsf{Trusted}$ or $f_k \in \mathbf{f}$ has variables not in $\mathbf{u}^{\mathsf{Fixed}}$, then abort and output $\perp$.
2. Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{f}^{\mathsf{Trusted}})$. For any $i \in \mathcal{I}$ and $j \leq |\mathbf{s}_i|$, let $S_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$. Below we let F_k be a variable for a pairing group element in $\mathbb{G}_{\alpha_k}$, which will be assigned a value later from a challenge for a PPE problem, and will make part of the PPE C.
3. If there exists a constant vector $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{s}_T|})$ with entries in $\mathbb{Z}_p$ such that $f_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j]$, then output the PPE

$$C := \begin{cases} e(F_k, g_2) = \prod_{j=1}^{|\mathbf{s}_T|} S_T[j]^{a_j} & \text{if } \alpha_k = 1 \\ e(g_1, F_k) = \prod_{j=1}^{|\mathbf{s}_T|} S_T[j]^{a_j} & \text{if } \alpha_k = 2 \\ F_k = \prod_{j=1}^{|\mathbf{s}_T|} S_T[j]^{a_j} & \text{if } \alpha_k = T \end{cases}$$

and PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted} \cup \{k\})$, else output $\perp$. Note that computing such a coefficient vector $\mathbf{a}$ reduces to checking if the polynomial $\mathbf{0}$ belongs to the span of polynomials $\mathbf{s}_T \cup \{f_k\}$.

Figure 2: Procedure for moving certain elements with all Fixed variables to Trusted.

Lemma 4.1 (Correctness of Rule 1)**.** *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ *be a PPE problem instance as in Definition 3.1 and let* $k \in [1, m]$*. Suppose* $\perp \neq (C, \Pi') = \mathsf{Rule1}(\Pi, k)$*. Then:*

1. Π' *is a PPE problem instance as in Definition 3.1 and*
2. *for every testing set* $T \in Q_{\Pi'}$*, it holds that* $(T \cup \{C\}) \in Q_{\Pi}$*.*

Proof. We have that $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$. Since $\mathsf{Rule1}(\Pi, k) = (C, \Pi') \neq \perp$, we know that the rule was successfully applied, where $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted} \cup \{k\})$.

Condition 1: we observe that Π' will also satisfy Definition 3.1, where the only non-trivial observation is that we must show that if $i \in \mathsf{Trusted} \cup \{k\}$, then f_i is a multivariate polynomial over the set of variables in $\mathbf{u}^{\mathsf{Fixed}}$ is maintained. This follows from the fact that a necessary condition for Rule1 to move k to Trusted is that f_k only has variables in $\mathbf{u}^{\mathsf{Fixed}}$.

Condition 2: let T be any testing set for Π'. By Definition 3.3, T is a set of pairing product equations such that each equation in $\{T_1, \ldots, T_w\} = T$ is simultaneously satisfied for all YES challenges, and at least one equation in T is not satisfied for all NO challenges. Recall there are no conditions on the behavior for INVALID challenges.

We now argue by contradiction that if $T \cup \{C\}$ is not a testing set for Π, then T cannot be a testing set for Π'. Since $T \cup \{C\}$ is not a testing set for Π, then either:

- Case 1: There exists a YES challenge $\mathbf{F}$ for Π such that at least one equation in $T \cup \{C\}$ is not satisfied, or
- Case 2: There exists a NO challenge $\mathbf{F}$ for Π such that all equations in $T \cup \{C\}$ are simultaneously satisfied.

We now analyze each of these cases.

In Case 1, we know that at least one equation in $T \cup \{C\}$ is not satisfied by challenge $\mathbf{F}$. We take this in two subcases. First, suppose that T contains an unsatisfied equation. This means that $\mathbf{F}$ is also a YES challenge for Π' (it can use the same settings for the variables), but for which one equation of T is not satisfied. This contradicts the starting assumption that T was a testing set for Π'. Second, suppose that all equations of T are satisfied, but that the equation C is not. By definition of being a YES challenge, we

know there exists an assignment to the variables $\mathbf{u}$ such that $F_i = g_{\alpha_i}^{f_i(\mathbf{u})}$ for all i. Equation C tests that F_k is equal to $g_{\alpha_i}^{f_i(\mathbf{u})}$, thus this equation being false contradicts the fact the $\mathbf{F}$ was a YES challenge.

In Case 2, since $\mathbf{F}$ is a NO challenge for Π where all equations in $T \cup \{C\}$ are simultaneously satisfied, then $\mathbf{F}$ is also a NO challenge for Π' where all equations in T are simultaneously satisfied. We argue this as follows. By Definition 3.2 of a NO challenge for Π, there exists an assignment to $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}_p^n$ such that for all $i \in \mathsf{Trusted}$, $F_i = g_{\alpha_k}^{f_k(\mathbf{u})}$. To convert this to a NO challenge for Π', we also need to show that $F_k = g_{\alpha_k}^{f_k(\mathbf{u})}$ for this same assignment $\mathbf{u}$. This follows from the fact that PPE C is satisfied by this challenge and that C explicitly tests that F_k is computed this way, possibly with respect to an equivalent polynomial for $f_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j]$. Now since $\mathbf{F}$ is NO challenge for Π', it remains to see how it performs with respect to the set T. However, since all equations in $T \cup \{C\}$ are satisfied by this challenge $\mathbf{F}$, then all equations in T are as well. This contradicts the original assumption that T was a testing set for Π'. ∎

4.2.2 Rule 2: Move element to Trusted by fixing an un-Fixed variable of form $v \cdot u_j^d$

Rule 2 is described in Figure 3. Given a PPE problem $\Pi = (\mathbb{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ and indices $j \in [1, n]$ and $k \in [1, m]$, Rule 2 can possibly be applied if $j \notin \mathsf{Fixed}$, $k \notin \mathsf{Trusted}$ and the polynomial $f_k \in \mathbf{f}$ is of the form $c \cdot u_j^d + h$, where the variable $u_j \in \mathbf{u}$, the polynomial h contains only variables in $\mathbf{u}^{\mathsf{Fixed}}$, constant $c \in \mathbb{Z}_p^*$, and constant $d \in \mathbb{Z}_p$ s.t. d is relatively prime to $p - 1$.

Description of Rule 2

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ and integers $j \in [1, n]$ and $k \in [1, m]$.
Output: A PPE problem Π' or $\perp$ (meaning could not apply the rule).
Steps of $\mathsf{Rule2}(\Pi)$:

1. If polynomial $f_k \in \mathbf{f}$ is of the form $c \cdot u_j^d + h$, where
   - $j \notin \mathsf{Fixed}$,
   - $k \notin \mathsf{Trusted}$,
   - the variable $u_j \in \mathbf{u}$,
   - the polynomial h contains only variables in $\mathbf{u}^{\mathsf{Fixed}}$,
   - the constant $c \in \mathbb{Z}_p^*$ and
   - the constant $d \in \mathbb{Z}_p$ is relatively prime to $p - 1$

   then output the PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed} \cup \{j\}, \mathsf{Trusted} \cup \{k\})$.
2. Otherwise, output $\perp$.

Figure 3: Procedure for moving certain elements to Trusted by fixing an un-Fixed variable

Let $\perp \neq \Pi' = \mathsf{Rule2}(\Pi, j, k)$. We now prove that a testing set for Π' is also a testing set for Π, which ensures that Rule 2 does not "flip" a non-testable PPE problem into a testable one.

Lemma 4.2 (Correctness of Rule 2)**.** *Let $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ be a PPE problem instance as in Definition 3.1, $j \in [1, n]$ and $k \in [1, m]$. Suppose $\perp \neq \Pi' = \mathsf{Rule2}(\Pi, j, k)$. Then:*

1. *Π' is a PPE problem instance as in Definition 3.1 and*
2. *for every testing set $T \in Q_{\Pi'}$, it holds that $T \in Q_{\Pi}$.*

Proof. We have that $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$. Since $\mathsf{Rule1}(\Pi, j, k) = \Pi' \neq \perp$, we know that the rule was successfully applied, where $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed} \cup \{j\}, \mathsf{Trusted} \cup \{k\})$.

Condition 1: we observe that Π' will also satisfy Definition 3.1, where the only non-trivial observation is that we must show that if $i \in \mathsf{Trusted} \cup \{k\}$, then f_i is a multivariate polynomial over the set of variables in $\mathbf{u}^{\mathsf{Fixed} \cup \{j\}}$. This follows from the fact that a necessary condition for Rule2 to move k to Trusted is that f_k only has variables in $\mathbf{u}^{\mathsf{Fixed} \cup \{j\}}$ and that j is simultaneously moved to Fixed.

Condition 2: the PPE problems Π and Π' differ only in their last two items: the Fixed and Trusted sets, where the Π' sets have the additional elements $\{j\}$ and $\{k\}$ respectively. As the definition of an YES challenge has no dependence on Trusted and Fixed sets, each YES challenge for Π is also an YES challenge for Π' and vice versa. Since T is a testing set for Π', each equation in T is simultaneously satisfied for all YES challenges of Π', and therefore satisfied for all YES challenges of Π.

Similarly, we argue that any NO challenge for Π is also a NO challenge for Π', meaning that at least one equation in T is not satisfied in both cases. Consider any NO challenge $\mathbf{F}$ for the PPE problem Π. By definition, $\mathbf{F}$ is not a YES challenge for Π (or, by the above, for Π'), and there exists an assignment of $\mathbf{u}^* \in \mathbb{Z}_p^n$ such that $F_i = g_{\alpha_i}^{f_i(\mathbf{u}^*)}\ \forall i \in \mathsf{Trusted}$.

We want to show that $F_k = g_{\alpha_k}^{f_k(\mathbf{u}^*)}$. Since $\mathsf{Rule2}(\Pi, j, k) \neq \perp$, we know that the polynomial f_k was of the form $c \cdot u_j^d + h$, according to the constraints of Rule2, where $j \notin \mathsf{Fixed}$. Thus, for this setting of F_k in the challenge $\mathbf{F}$, there exists only one setting of the variable $u_j \in \mathbb{Z}_p$ that is consistent with F_k being derived via the polynomial f_k and the settings of $u_i \in \mathbf{u}^*$ for all $i \in \mathsf{Fixed}$. Let $F_k = g^y$ for some $y \in \mathbb{Z}_p$. Then we have that:

$$u_j = (\frac{y-h}{c} \mod p)^{1/d \mod (p-1)}.$$

There is a unique solution to the above since d is relatively prime to $p-1$. Recall that h is derived over the set of variables in $\mathbf{u}^{\mathsf{Fixed}}$.

By Definition 3.1 of a PPE Problem, we have that if $i \in \mathsf{Trusted}$, then f_i is a multivariate polynomial over the set of variables in $\mathbf{u}^{\mathsf{Fixed}}$. We observe that Rule2 preserves this condition by only moving a polynomial's index to Trusted if it over the set of variables in $\mathbf{u}^{\mathsf{Fixed}}$ and the variable u_j which it simultaneously moves to Fixed.

Thus, for the same setting of variables $\mathbf{u}^* \in \mathbb{Z}_p^n$, it holds that $F_i = g_{\alpha_i}^{f_i(\mathbf{u}^*)}\ \forall i \in (\mathsf{Trusted} \cup \{k\})$. This allows us to conclude that if $T \in Q_{\Pi'}$, then $T \in Q_{\Pi}$. ∎

4.3 Applying the Rules

We now show how to apply these rules in our main searching algorithm. When $\mathsf{QSearch}(\Pi)$ returns a testing set Q, we conclude Π is PPE testable. When the message unknown is returned, the algorithm failed to find a testing set. It does not, however, allow us to conclude anything about Π's PPE testability.

At a high-level, for input Π, if all elements are represented as Trusted, then $\mathsf{QSearch}(\Pi)$ returns the trivial testing set $\emptyset$ and the PPE problem is trivially PPE testable. Otherwise, the algorithm attempts to apply Rule1, which seeks to move an un-Trusted element into Trusted via a PPE C that can test it with respect to the other Trusted elements. If Rule1 can be applied, then the algorithm recurses on the PPE problem with one fewer un-Trusted elements and, if a testing set Q' for this "smaller" problem is found, it outputs the joint testing set $C \cup Q'$. If Rule1 cannot be applied, then the search algorithm tries to apply Rule2, which seeks to move an un-Trusted element into Trusted via fixing an un-Fixed variable. If Rule2 can be applied, then the algorithm again recurses on the smaller instance and, if a testing set for this instance is found, it outputs it likewise. We claim that while the order in which we attempt to apply Rule1 doesn't matter, the order for Rule2 possibly might; thus, our implementations will sometimes randomize the search order of the indices for this step (as opposed to the numerically increasing order we present here). We can explore other implementation options, such as first applying Rule2 to variables with the smallest d constants, etc.

Efficiency of QSearch. We now discuss about the asymptotic complexity of the QSearch algorithm. First observe that the size of the Trusted set increases by one with each recursive call to QSearch. Consequently, the function QSearch is called recursively at most m times. During each call, all the untrusted polynomials are scanned to check if any rule is applicable. For the purpose of this analysis, let us denote the size of a polynomial to be the total number of additions and multiplications involved in the normal form of the polynomial (e.g., the size of $x^2yz + 3z^3y^3$ is 5). Therefore, multiplying 2 polynomials of size s_1 and s_2 takes $O(s_1 s_2)$ time. Let the the maximum size of all polynomials $\mathbf{f}$ in the input be s. Executing Rule1 involves computing completion lists followed by checking if $\mathbf{0}$ lies in span of certain polynomials. We know that

Description of Main Algorithm for PPE Testing Set Search

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$.
Output: A PPE testing set Q or the message unknown.
Steps of $\mathsf{QSearch}(\Pi)$:

1. If $\mathsf{Trusted} = [1, m]$, then output the testing set $Q = \emptyset$.
2. For $k = 1$ to m,
   (a) Call $z = \mathsf{Rule1}(\Pi, k)$.
   (b) If $z = (C, \Pi') \neq \perp$, then
      i. call $Q' = \mathsf{QSearch}(\Pi')$ and
      ii. if $Q' \neq$ unknown, then output the testing set $Q = C \cup Q'$.
3. For $k = 1$ to m and $j = 1$ to n,
   (a) Call $\Pi' = \mathsf{Rule2}(\Pi, j, k)$.
   (b) If $\Pi' \neq \perp$, then
      i. call $Q = \mathsf{QSearch}(\Pi')$ and
      ii. if $Q \neq$ unknown, then output the testing set Q.
4. Otherwise output unknown.

Figure 4: Recursive procedure for searching for a PPE Testing Set

computing completion lists of m polynomials involves $O(m^2)$ polynomial multiplications taking $O(m^2 \cdot s^2)$ time. Checking if $\mathbf{0}$ lies in span of $O(m^2)$ polynomials (number of polynomials in completion lists) involves solving a system of $O(m^2 \cdot s)$ linear equations (upper bound on number of monomials in completion list) each of size $O(m^2)$, which takes at most $O((m^2 \cdot s)^\omega)$ time, where n^ω is the complexity of multiplying two $n \times n$ matrices. (Current best known value of ω is 2.3728639 [27]). Consequently, the time taken to execute Rule1 on an untrusted polynomial is $O(m^{2\omega} \cdot s^\omega)$. The time taken to execute Rule2 on a polynomial f is only linear in terms of the size of f. Consequently, every recursive call takes $O(m^{2\omega+1} \cdot s^\omega)$ time to check for both the rules on m polynomials. As QSearch involves at most m recursive calls, the total time taken is at most $O(m^{2\omega+2} \cdot s^\omega)$.

Theorem 4.1 (Correctness of Testing Set from Algorithm in Figure 4)**.** *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ *be a PPE problem instance as in Definition 3.1. Let* $Q = \mathsf{QSearch}(\Pi)$*. If* $Q \neq$ unknown*, then* Q *is a testing set for* Π*.*

Proof. We want to show that if $\mathsf{QSearch}(\Pi) = Q \neq$ unknown, then Q is a testing set for Π. We do this by induction.

Base Case: When $\mathsf{Trusted} = [1, m]$, then $\mathsf{QSearch}(\Pi) = \emptyset$. In this case, all elements are Trusted; thus all PPE challenges for Π are either YES or INVALID challenges. The emptyset $\emptyset$ trivially satisfies Definition 3.3.

Induction Step: For any $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}', \mathsf{Trusted}')$ where $\mathsf{Trusted}' \subseteq [1, m]$ and Q' is a testing set for Π', we prove that:

1. If $\Pi'' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}', \mathsf{Trusted}'' = \mathsf{Trusted}' \setminus \{k\})$ and $(C, \Pi') = \mathsf{Rule1}(\Pi'', k)$, for some $k \in [1, m]$, then $Q'' = C \cup Q'$ is a testing set for Π''.
2. If $\Pi'' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}'' = \mathsf{Fixed}' \setminus \{j\}, \mathsf{Trusted}'' = \mathsf{Trusted}' \setminus \{k\})$ and $(C, \Pi') = \mathsf{Rule2}(\Pi'', k)$, for some $k \in [1, m]$, then $Q'' = Q'$ is a testing set for Π''.

We have two cases.

Case 1: Since Q' is a testing set for Π', then we know it tests all $\ell = m - |\mathsf{Trusted}'|$ untrusted elements in Π'. We need to argue that $Q'' = C \cup Q'$ tests all $\ell + 1 = m - |\mathsf{Trusted}''| = m - |\mathsf{Trusted}'| + 1$ untrusted elements in Π''. In particular, the sole element they differ in is the element represented by index k. By

inspection of Rule1, we see that the PPE C allows for testing this element k. Thus, if Q' is a set of PPEs testing all untrusted elements except k and C is a PPE testing element k, then together they form a testing set for all untrusted elements in Π''.

Case 2: Since Q' is a testing set for Π', then we know it tests all $\ell = m - |\mathsf{Trusted}'|$ untrusted elements in Π'. We need to argue that $Q'' = Q'$ tests all $\ell + 1 = m - |\mathsf{Trusted}''| = m - |\mathsf{Trusted}'| + 1$ untrusted elements in Π''. In particular, the sole element they differ in is the element represented by index k. Since Rule2 was successfully applied, element k was moved to the trusted set, because it contained an unfixed variable, whose value was fixed by the move. Since the variable was unfixed but is now being fixed by the move, no PPE is required to test this element as a trusted element. All future uses of this variable will now be tested against its fixed value. Thus, no new PPEs are required for the execution of Rule2 and $Q'' = Q'$ is a testing set for Π''.

Argument Summary: Since we have shown that $\mathsf{QSearch}(\Pi) \neq \mathsf{unknown}$ returns a testing set properly for the base case where all elements are trusted and we have shown the inductive step that for all Π', Π'' which differ by only one trusted element, according to the relationships of Rule1 or Rule2 called by QSearch, that QSearch correctly derives a testing set for Π'' from a testing set for Π', then by the principle of induction we have shown that any testing set output by QSearch is correct.

■

Corollary 4.1 (Correctness of PPE Testability from Algorithm in Figure 4)**.** *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Fixed}, \mathsf{Trusted})$ *be a PPE problem instance as in Definition 3.1. Let* $Q = \mathsf{QSearch}(\Pi)$*. If* $Q \neq \mathsf{unknown}$*, then* Π *is PPE Testable.*

Proof. Follows directly from Theorem 4.1 and Definition 3.3. ■

5 Implementation and Case Studies

We now describe a new software tool, called AutoPPE, which implements the PPE searching algorithm presented in Figure 4. We ran AutoPPE on a number of IBE, ABE, VRF, signature schemes, and other type of pairing-based public/private parameters, including some that are PPE testable and some that are provably not PPE testable. We report on the design, results and performance of AutoPPE in this section.

5.1 AutoPPE Implementation

We implemented the AutoPPE tool using Ocaml version 4.02.3. We utilized the parsing tools and data structures (to store polynomials) from the Generic Group Analyzer (GGA)[2]. We used the SageMath package[3] to solve systems of linear equations. We implemented the remaining logic ourselves.

AutoPPE takes as input pairing information (such as the Type I, II or III), a set of fixed/unfixed variables, and a set of trusted/untrusted polynomials along with their group identifiers. (While this is a slightly different format than we used in Definition 3.1, we stress that it is the same information.) In addition, the tool optionally takes as input information that allows the tool to help the user encode some cryptosystem parameters as a PPE problem instance. In particular, all trusted and untrusted elements (represented by polynomials) are bilinear group elements in $\mathbb{G}_1, \mathbb{G}_2$ or $\mathbb{G}_T$ and Definition 3.1 does not allow including an element in $\mathbb{Z}_p$ in either set. However, since it is not uncommon for schemes to contain elements in the $\mathbb{Z}_p$ domain as part of their public or private parameters, we implemented a workaround for those schemes as described in Section 5.2. The tool runs the Figure 4 search algorithm, with a few optimizations detailed in Section 5.5 and a few limitations detailed in Section 5.6, and outputs either a set of PPEs or the special symbol unknown. After obtaining PPEs by running QSearch algorithm, the tool further runs an algorithm similar to [7] to produce equivalent PPEs which are more efficient to check.

[2] https://github.com/generic-group-analyzer/gga
[3] https://www.sagemath.org/

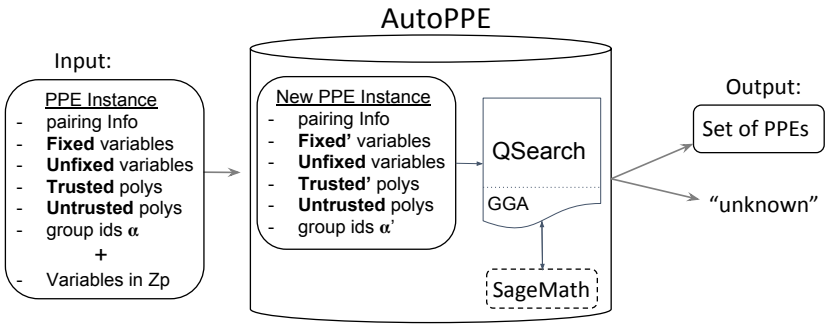

Figure 5: The workflow of the AutoPPE tool which follows the logic in Figure 4. It takes as input an intial PPE problem instance along with some additional information (i.e., variables in $\mathbb{Z}_p$) that help the user encode a given pairing-based scheme into a proper PPE problem instance, as we explain in the text. The tool utilizes and adapted portions of existing tools such as the Generic Group Analyzer (GGA) for handling polynomials and completion sets and the SageMath package for solving systems of linear equations. The output is either a set of Pairing Product Equations (PPEs), indicating that the instance is PPE Testable, or the special symbol unknown.

The source code for AutoPPE comprises about 3K lines of Ocaml code, and the input description of each pairing based scheme we tested consists of less than 10 lines of code. The ease of converting a given pairing based scheme into the input format for AutoPPE makes the tool highly practical and useful. We plan to make AutoPPE publicly available as open source code at `https://github.com/JHUISI/auto-tools`.

5.2 Encoding "Well-formedness" of Cryptosystem Parameters as a PPE Testability Problem

In this subsection, we describe how to look at the public-private parameters of a pairing-based cryptosystem and then encode this as a PPE problem instance. Typically, the objective is to test that the private parameters are "well formed" with respect to the public parameters, where the definition of being "well formed" depends on the application. Let's take identity-based encryption (IBE) as our starting example. For an IBE scheme $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, there are a number of different applications (see the discussion in Section 1) where one wants an efficient deterministic procedure (based on PPEs) that takes as input the public parameters pp, an identity id and a purported private key S, and verifies whether S is a possible output of the KeyGen algorithm with respect to pp and id. Recall that the critical point of our work is *discovering* whether a scheme's parameters can be verified in this way or not.

We now formulate the problem of determining well-formedness of a pairing-based IBE secret key as an instance of the PPE Testability problem. Suppose for the given IBE scheme on group structure $\mathcal{G}$, the public key is of the form $(g_{\alpha_1}^{f_1}, \ldots, g_{\alpha_k}^{f_k})$ and the secret key for an identity is of the form $(g_{\alpha_{k+1}}^{f_{k+1}}, \ldots, g_{\alpha_m}^{f_m})$, where $\alpha_i \in \{1, 2, T\} \forall i$, $\{f_1, \ldots, f_k\}$ are polynomials on variables $\{u_1, u_2, \ldots, u_t\}$ and $\{f_{k+1}, \ldots, f_m\}$ are polynomials on variables $\{u_1, u_2, \ldots u_n\}$. We formulate the corresponding PPE problem as $(\mathcal{G}, n, m, \mathbf{u} = \{u_1, u_2, \ldots, u_n\}, \mathbf{f} = \{f_1, f_2, \ldots, f_m\}, \boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}, \mathsf{Trusted} = [1, k], \mathsf{Fixed} = [1, t])$.

Although this encoding seems quite simple, many IBE constructions deviate from this form in several ways. We now describe several insights into converting a given IBE scheme into the above form.

(1) When multiple group elements are sampled randomly in a scheme, we first normalize the scheme by using a single generator for each group and replacing every random sampling of a group element by g^v, where the g is generator of the group and v is a fresh variable randomly sampled from $\mathbb{Z}_p$.

(2) Many constructions such as Boyen-Waters [24] and Waters dual system [47] include identity id or $Hash(\mathsf{id})$ as part of the private key. In general, constructions which include variables $\{v_1, v_2, \ldots, v_s\}$ in $\mathbb{Z}_p$ as part of public/secret key can be reformulated into a PPE problem instance with the following modifica-

tions. Expand the Trusted set by including $f_i \cdot \mathsf{poly}(\mathsf{v_1}, \mathsf{v_2}, \ldots, \mathsf{v_s})$ with group identitifer α_i for every trusted polynomial f_i and every polynomial poly() of degree at most d. We also include the variables $\{v_1, v_2, \ldots, v_s\}$ as part of the Fixed set. The parameter d can be easily configured in the tool. We used $d = 1$ for our case studies and observed that it is sufficient for all the schemes that we tested.

(3) The Boneh-Franklin [22] and Gentry-Silverberg [29] constructions use a hash function H that hashes identities to a group element. In this case, we reformulate the scheme by replacing $H(\mathsf{id})$ with g_α^h for an appropriate $\alpha \in \mathcal{I}$ and a fresh variable h.

(4) The Boneh-Boyen [20] construction hashes identity into bit string $H(\mathsf{id}) = h_1||h_2||h_3||\cdots||h_k$, where k is the length of the bit string. The Waters/Naccache [42] construction hashes identity into blocks of bit strings $H(\mathsf{id}) = h_1||h_2||h_3||\cdots||h_k$, where each h_i is a bit string block and k is the number of blocks in $H(\mathsf{id})$. In either case, we first reformulate the problem by considering each h_i as a separate variable in $\mathbb{Z}_p$ and including it as a part of the secret key. We then reduce it to the PPE Testability problem as described earlier. Note that, this method results in a significant blowup in the number of polynomials in the input and can be tested efficiently only for modest values of n. However, the output PPEs can be manually extended to higher values of n by identifying a pattern.

Using the above encoding approaches, we tested 8 pairing-based IBE schemes for well-formedness of the private key and our tool was able to quickly output a testing set for all of the schemes which are testable.

We now look beyond IBE schemes. A signature scheme $\mathsf{SIG} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ is said to be well-formed if there exists an efficient deterministic procedure to verify that a given signature is a valid (possible) signature w.r.t. given message and public key. Similarly, a Verifiable Random Function (VRF) scheme $\mathsf{VRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ is said to be well-formed if there exists an efficient deterministic method to test that a given VRF output and proof are valid w.r.t. given verification key and input. Analogously, a Ciphertext-Policy Attribute Based Encryption (CP-ABE) scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to have well-formed secret key if there exists an efficient deterministic way to check that a given ABE secret key is valid w.r.t. given public key and attributes. Testing whether a given pairing-based Signature/VRF/CP-ABE scheme is well-formed can be reformulated as a PPE testability problem analogous to the IBE case described above.

5.3 A Detailed Example for the Waters05 IBE

Before presenting all our cases studies in Section 5.4, we'd like to walk the reader through one detailed example. Let us consider the Waters05 IBE scheme [45] with the Naccache Optimization [42]. We would like to check if the private key for an identity is PPE Testable given the public parameters and the identity. As mentioned in the introduction, an IBE scheme with "private key" PPE Testability immediately implies a signature scheme with deterministic verification. Moreover, an IBE scheme with "private key" PPE Testability, and a few other properties, admits an adaptive oblivious transfer scheme [32]. For the sake of completeness, we recall this popular construction in Appendix A.

The input file for the tool is presented in Figure 6. For space reasons, we choose to illustrate this with a toy example of 4 as the identity block size; in practice one would likely use 8 or 32.[4] The pairing information is specified using the line `maps G1*G1->GT`, which denotes a Type I pairing. Alternately, a Type II pairing could be specified by `maps G1*G2->GT, isos G1->G2`, and a Type III pairing could be specified by `maps G1*G2->GT`. In order to test the for well-formedness of an IBE private key, the public parameter elements (Trusted set) along with their group identifiers are specified by `trusted_polys [_] in G_`, and the private key elements for an identity ($\overline{\mathsf{Trusted}}$ set) along with their group identifiers are specified using `untrusted_polys[_] in G_`. Every polynomial should be specified along with a unique identifier which will be used to output the PPEs in a compact form. The variables sampled in the Setup phase (Fixed set) are specified using `fixed_vars [_].` and the variables sampled during the KeyGen phase ($\overline{\mathsf{Fixed}}$ set) are specified using `unfixed_vars [_]`. The IBE construction hashes identity id into blocks of bit strings, which can be

[4]Specifically, one would likely choose to support arbitrary-length identity strings by first hashing them to 256 bits using SHA-256 and then applying the Naccache optimization [42] of dividing these 256-bit identities into eight 32-bit blocks or thirty-two 8-bit blocks.

Input File Example

```
(*Waters05 IBE scheme with Nacaccche optimization for symmetric pairings with number of blocks = 4*)
maps G1 * G1 ->GT.
fixed_vars [alpha, beta, u, u1, u2, u3, u4].
unfixed_vars [r].
(*Variables corresponding to blocks of H(id)*)
Zp_vars [v1, v2, v3, v4, v5].

(*Public key*)
trusted_polys [F1 = alpha, F2 = beta, F3 = u, F4 = u1, F5 = u2, F6 = u3, F7 = u4] in G1.

(*Secret key for an identity*)
untrusted_polys [F8 = r, F9 = alpha*beta + (u + v1*u1 + v2*u2 + v3*u3 + v4*u4)*r] in G1.
```

Figure 6: Input file for Waters05 IBE scheme with Nacaccche Optimization.

treated as elements in $\mathbb{Z}_p$ for our purposes. Each of the blocks is identified by a separate variable, and are specified using `Zp_vars [_]`. Comments in the input file can be specified using `(*....*)`.

The output of the tool on the above input is presented in Figure 7. The tool first converts the input specification to a PPE instance by multiplying every variable specified in `Zp_vars[_]` with every trusted polynomial and including them in trusted set. This expands the Trusted set from 9 polynomials (including the identity polynomials internally added by our tool) to 45 polynomials which are printed in the output. The tool later on applies the QSearch algorithm and outputs the PPEs in terms of the unique identifiers specified for each polynomial. Note that the tool also optimizes the PPEs to minimize the number of pairings used in the PPE. Further optimization can be achieved using AutoBatch tool [7, 8][5] which can batch many PPEs into few PPEs.

5.4 Case Studies

We evaluated AutoPPE on various types of pairing-based schemes using a MacBook Pro 2015 laptop with 2.7GHz Intel Core i5 processor and 8GB 1867MHz DDR3 RAM. We present the results along with average execution times over 10 runs in Table 1. In Appendix B, we include more details about the input and output of AutoPPE on some test schemes. We observe that the tool outputs a testing set for most of the standard schemes which are testable within a few seconds.

We note that in our implementation, we simplify checking whether the constant d is relatively prime to $p-1$ in Rule2, by checking whether d is a small prime ($d \in \{1, 3, 5, 7, 11\}$). We made this simplification is because none of the schemes we encountered include a polynomial with degree $d > 2$ for an unfixed variable.

In order to mimic the schemes presented in the papers as they are, we tested most of the schemes in the Type I setting. To demonstrate the flexibility of the tool, we also translated several of these schemes into the Type III setting[6]). The Waters dual system IBE [47] is not PPE testable (see Section 1) and our tool (correctly) output unknown (see the full output in Appendix B). The Boyen-Waters anonymous IBE [24] and the Dodis VRF [26] appear not to be PPE testable (see Section 5.6) and our tool also output unknown for these.

The introduction motivated this problem by showing a connection between PPE testability for an IBE scheme and its suitability for use in blind and/or accountable authority IBE systems. We remark that we tested several such IBE schemes as part of our case study, including Boneh-Boyen [20], Waters [45] and Naccache [42] (which were employed in [32] to leverage this property to build OT).

[5] https://github.com/JHUISI/auto-tools/tree/master/auto_batch

[6] We encoded elements into $\mathbb{G}_1$ or $\mathbb{G}_2$ using the position they appear in the original papers. There is no guarantee this Type I to Type III translation maintains the scheme's security, but we are only concerned here with deriving test cases and we wanted to keep our translations easy for the reader to rediscover. For secure pairing translation methods, see [2, 5, 3].

Scheme	Pairing	Type	PPE Testability	Tool's Output	Execution Time
Boneh-Franklin01 ([22])	Type I	IBE	Testable	Testable	1.90s
Gentry-Silverberg02 ([29])	Type I	IBE	Testable	Testable	2.94s
Boneh-Boyen04a ([19]) ($l = 160$)	Type I	HIBE	Testable	Testable	5.55s
Boneh-Boyen04b ([20]) ($\|H(\mathsf{id})\| = 16$)	Type I	IBE	Testable	Testable	9.23s
Waters05 ([45]) ($\|H(\mathsf{id})\| = 160$)*	Type I	IBE	Testable	Testable	6.91s
Waters05 ([45]) ($\|H(\mathsf{id})\| = 16$)	Type I	IBE	Testable	Testable	3.87s
Naccache05 ([42]) ($\mathcal{B}(H(\mathsf{id})) = 8$)	Type I	IBE	Testable	Testable	1.69s
Naccache05 ([42]) ($\mathcal{B}(H(\mathsf{id})) = 8$)	Type III	IBE	Testable	Testable	1.66s
BBG05 ([21]) ($l = 8$)	Type I	HIBE	Testable	Testable	10.96s
Boyen-Waters06 ([24])	Type I	Anon-IBE	see Section 5.6	Unknown	0.0008s
Waters09 ([47])	Type I	IBE	Not Testable	Unknown	1.57s
BLS01 ([23])	Type I	Signature	Testable	Testable	1.69s
CL04 Scheme A ([25])	Type I	Signature	Testable	Testable	3.12s
CL04 Scheme B ([25])	Type I	Signature	Testable	Testable	6.21s
CL04 Scheme B ([25])	Type III	Signature	Testable	Testable	6.53s
CL04 Scheme C ([25]) ($\mathcal{B}(\mathsf{msg}) = 8$)	Type I	Signature	Testable	Testable	25.81s
Dodis03 ([26]) ($\|C(x)\| = 6$)	Type I	VRF	see Section 5.6	Unknown	0.18s
Dodis03 ([26]) ($\|C(x)\| = 6$)	Type III	VRF	see Section 5.6	Unknown	0.12s
Lys02 ([40]) ($\|C(x)\| = 5$)	Type I	VRF	Testable	Testable	8.78s
Lys02 ([40]) ($\|C(x)\| = 5$)	Type III	VRF	Testable	Testable	9.10s
Jager15 ([37]) ($\|H(x)\| = 4$)	Type I	VRF	Testable	Testable	9.11s
Jager15 ([37]) ($\|H(x)\| = 4$)	Type III	VRF	Testable	Testable	9.98s
RW13 ([44]) ($a = 60$)	Type I	CP-ABE	Testable	Testable	222.75s
RW13 ([44]) ($a = 60$)	Type III	CP-ABE	Testable	Testable	222.43s
100-DDH	Type I	Custom	Testable	Testable	1.77s
100-DBDH	Type I	Custom	Not Testable	Unknown	0.16s

Table 1: The output of AutoPPE on various PPE testability problems. Here, l represents the number of delegation levels in a HIBE scheme, $|H(\mathsf{id})|$ denotes the length of the hash of identity id, $\mathcal{B}(H(\mathsf{id}))$ denotes the number of blocks in the hash of identity id, $\mathcal{B}(\mathsf{msg})$ denotes the number of blocks in message msg, $|C(x)|$ denotes the length of encoding of input x, $|H(x)|$ denotes the length of encoding of input x and a denotes the number of attributes. "*" indicates that an optimized encoding mechanism is used to account for elements in $\mathbb{Z}_p$. The execution time is mentioned in seconds.

Output of the Tool

```
F0 = 1 in G1       F0 = 1 in GT       F1 = alpha in G1       F2 = beta in G1       F3 = u in G1       F4 = u1 in G1
F5 = u2 in G1      F6 = u3 in G1      F7 = u4 in G1      F8 = r in G1
F9 = alpha*beta + r*u + r*u1*v1 + r*u2*v2 + r*u3*v3 + r*u4*v4 in G1
F10 = v1 in G1     F11 = v2 in G1     F12 = v3 in G1     F13 = v4 in G1
F14 = v1 in GT     F15 = v2 in GT     F16 = v3 in GT     F17 = v4 in GT
F18 = alpha*v1 in G1     F19 = alpha*v2 in G1     F20 = alpha*v3 in G1     F21 = alpha*v4 in G1
F22 = beta*v1 in G1      F23 = beta*v2 in G1      F24 = beta*v3 in G1      F25 = beta*v4 in G1
F26 = u*v1 in G1      F27 = u*v2 in G1      F28 = u*v3 in G1      F29 = u*v4 in G1
F30 = u1*v1 in G1      F31 = u1*v2 in G1      F32 = u1*v3 in G1      F33 = u1*v4 in G1
F34 = u2*v1 in G1      F35 = u2*v2 in G1      F36 = u2*v3 in G1      F37 = u2*v4 in G1
F38 = u3*v1 in G1      F39 = u3*v2 in G1      F40 = u3*v3 in G1      F41 = u3*v4 in G1
F42 = u4*v1 in G1      F43 = u4*v2 in G1      F44 = u4*v3 in G1      F45 = u4*v4 in G1

Processing untrusted polynomial F8 = r by rule2
F8 moved to trusted set and r moved to fixed set by rule 2

Processing untrusted polynomial F9 by rule1
Naive PPE e(F9,F0) = e(F1,F2) * e(F3,F8) * e(F8,F30) * e(F8,F35) * e(F8,F40) * e(F8,F45)
Optimized PPE e(F9,F0) = e(F1,F2)*e(F3*F30*F35*F40*F45,F8)
F9 moved to trusted set by rule 1

Execution time : 2.578486s
PPEs : e(F9,F0) = e(F1,F2)*e(F3*F30*F35*F40*F45,F8)
Ouptut : PPE Testable
```

Figure 7: Output of AutoPPE for Waters05 IBE scheme with Nacacche Optimization

In the [26, 40, 37] VRF schemes, the input is encoded as a bit string, which is treated as a vector of $\mathbb{Z}_p$ variables by our tool. We observe that the size of the polynomials in these schemes grow exponentially in size with respect to the length of encoding of the input. Consequently, we tested these schemes only with a short length encoding. However, for these schemes, we observe that the PPEs have a clear pattern which can be extrapolated to input encodings of arbitrary length.

In Section 5.2, we described a method to encode schemes which output elements in $\mathbb{Z}_p$ as part of their trusted or untrusted parameters as a PPE Testability problem. The naïve method of reformulating such schemes blows up the size of the trusted set and is inefficient when there are large number of elements in $\mathbb{Z}_p$. However, for a few problems we can improve the run time by including only a subset of these additional polynomials in the trusted set; one can always try a smaller set first and then expand the input iteratively. We demonstrate this using the Waters05 [45] example, which hashes identities to 160 bit strings and as a result would blow up the size of the trusted set to $O(160^2)$ polynomials when encoded naïvely. We improve upon the naïve method by including only 480 polynomials in the trusted set and thereby achieving significantly faster run times.

We also tested our tool on a few custom examples with 100+ elements in them. In the 100-DDH example with Type I pairings, the trusted set contains polynomials $\{a_1, a_2, \cdots, a_{50}\}$ in group $\mathbb{G}_1$, the untrusted set contains polynomials $\{a_{51}, a_{52}, \cdots, a_{100}, b\}$ in group $\mathbb{G}_1$ and the polynomial $(a_1 + a_2 + \cdots + a_{100}) * b$ in group $\mathbb{G}_T$. Clearly, this problem can be tested using the PPE $\mathsf{g_T}^{(a_1+a_2+\cdots+a_{100})*b} = e(\prod_{i=1}^{100} \mathsf{g_1}^{a_i}, \mathsf{g_1}^b)$. In the 100-DBDH example with Type I pairings, the trusted set contains $\{a_1, a_2, \cdots, a_{50}\}$ in group $\mathbb{G}_1$, the untrusted set contains polynomials $\{a_{51}, a_{52}, \cdots, a_{100}, b, c\}$ in group $\mathbb{G}_1$ and the polynomial $(a_1 + a_2 + \cdots + a_{100}) * b * c$ in group $\mathbb{G}_T$. This problem is not PPE Testable under the Decisional Bilinear Diffie-Hellman (DBDH) assumption as it involves deciding a DBDH instance.

5.5 Optimizations

The QSearch algorithm discussed in Figure 4 has a high time complexity. It is particularly unacceptable for problems which include lot of elements in the $\mathbb{Z}_p$ domain due to the blowup of the size of the trusted set and thereby the size of completion list. We therefore implemented a few optimizations which drastically improve the run time. We first observe that the completion lists get updated only by a few elements every time a polynomial is added to the trusted set. Consequently, the algorithm computes the completion list in an incremental manner instead of computing it from scratch each it checks for Rule 1. The completion list is updated each time a polynomial is added to the trusted set.

We further optimized the algorithm to find coefficients in Rule 1 by removing a subset of the polynomials that trivially have zero coefficient. When applying Rule 1, suppose a polynomial g in $\mathbf{s}_T \cup \{f_k\}$ has a monomial which is not present in any other polynomial, then trivially the coefficient g is zero when expressing $\mathbf{0}$ as a span of polynomials in $\mathbf{s}_T \cup \{f_k\}$.

5.6 Limitations and Open Problems

This work represents a meaningful first step in defining, understanding and automating the PPE testability of many well-known pairing cryptosystems. We now remark on some limitations of the tool that are exciting areas for future research.

(1) *Beyond Prime Order Pairings.* First, we restrict ourselves to pairing-based constructions with prime order groups. It would be interesting to extend the tool to composite-order pairings, e.g., [38, 39], and even RSA-based constructions. In constructions based on composite-order pairings, elements are sampled from various subgroups. Verifying the validity of untrusted terms involves testing whether the terms are in their designated subgroups. Our current model of representing a term with a polynomial may not be enough to verify such relationships.

(2) *Rational Polynomials and More.* Second, the tool doesn't work on schemes, such as Gentry's IBE [28] or Boneh-Boyen [19], which have group elements with exponents as rational polynomials (e.g., $g^{\frac{1}{p(\cdot)}}$) or schemes, such as Hohenberger-Waters [36], with polynomials with variable degree (e.g., g^{x^y}). While the GGA tool on which we built AutoPPE also does not handle rational polynomials, emerging new work by Ambrona et al. [9] does. While inspirational, it does not directly apply here. They work in the average case setting, and thus can ignore the negligible probability that an element is "undefined" (e.g., $g^{\frac{1}{p(\cdot)}}$ where $p(\cdot)$ evaluates to zero on a randomly chosen input). Here we focus on the worst case setting – where a powerful adversary such as an IBE Master Authority might try to pass off an ill-formed private key to a user using any such loophole – and we need a set of PPEs that can catch any ill-formed key.

(3) *Dependent Variables.* Third, the algorithm works only on schemes which sample all the variables independently. For example, the framework doesn't capture schemes which use hard-core predicate bits, schemes which use hash functions in complex ways (such as having g^x with $x = H(m||S)$, where S is another group element) [34] or schemes which sample two orthogonal vectors [43]. However, this seems to be more a limitation of what can be tested with PPEs rather than a limitation of this particular tool.

(4) *Unbounded PPEs.* Fourth, our scheme considers only constructions with concrete numbers of elements and parameters. One might consider extending this to the unbounded setting, as was done for the generic group assumption setting by Ambrona et al. [10].

(5) *On Relaxing PPE Testability.* Our automation outputs unknown on the Dodis VRF [26], which might seem surprising, since one might predict that the PPEs of the VRF verification algorithm would form a PPE testing set. In the case of this VRF, it does not *appear* to be PPE testable, even though its verification scheme is correct. Our Rule 1 does not apply because the PRF output $F \in \mathbb{G}_1$ being verified must be paired with an element $g_2^a \in \mathbb{G}_2$ (as opposed to just g_2) to be compared against other established values such as an $A \in \mathbb{G}_T$, i.e., an equation is checked of the form $e(F, g_2^a) = A$. The problem is that if $a = 0$ and $A = 1$, then $e(F, g_2^a) = A$ for *any* value of F, thus verifying nothing about F. In the scheme, a is chosen at random by a trusted party and thus it will be zero with only negligible probability, making this a non-issue for scheme security. However, our current formulation of PPE challenges and testability, in Definitions 3.2 and 3.3 respectively, requires that a testing set outputs the correct response for all challenges – which for

this scheme would include ones where $a = 0$, but in this case the VRF's verification equations do not appear to be sufficient. An interesting open direction could be to consider ways to relax our PPE testability notion to capture schemes such as Dodis [26], where one may be able to test most, but not all possible inputs.

(6) *PPE Circuits.* Lastly, and perhaps of most surprise to us, we look at what we learned from the Boyen-Waters anonymous IBE [24] example. This one again falls into a gray area – where we aren't actually sure if it is PPE testable or not. However, we found evidence leading us to *believe* that this scheme is: (a) not PPE testable under Definition 3.3, but is (b) potentially testable under a broader definition that would encompass outputting a "PPE circuit" rather than a conjunction of PPEs.

Let's dig in further. In an anonymous IBE system, the anonymity requirement is that the *ciphertext* does not reveal the identity of the recipient. Thus, there does not seem be anything inherent in such a system (unlike with dual system encryption) that would make the private key not be PPE testable. The AutoPPE tool outputs unknown for this case, and it quickly rejects it by finding that it cannot apply Rule2.

In Definition 3.3, we define testing sets to be a conjunction of PPEs. As a result, we needed to restrict the applicability of Rule2 only to polynomials of the form $f(\cdot) = f_1(\cdot) \cdot u^d + f_2(\cdot)$, where f_1 is a non-zero constant. In case f_1 is a non-constant polynomial on fixed variables, the untrusted polynomial f could be verified and moved to the trusted set, if we include conditional logic such as $(F_1 = I) \implies (F = F_2)$ in the testing set. Here, F is the formal challenge variable corresponding to the untrusted polynomial f, I is an identity element, F_1 and F_2 are expressions on formal variables corresponding to trusted polynomials which evaluate to $g_\alpha^{f_1}$ and $g_\alpha^{f_2}$ respectively for an appropriate group identifier α. Essentially, the rule states that in case f_1 evaluates to 0 for the given challenge, then f should evaluate to the value of f_2. In order to include such a rule, we will need to modify the definition of testing set to be a propositional logic on PPEs (or PPE circuit, rather than conjunction of PPEs). Extending the notion of PPE testability would appear to make the Boyen-Waters IBE scheme [24] PPE Testable, as its private key contains two polynomials of the form $f_1(\cdot) \cdot u_i + f_2(\cdot)$, where u_i is an unfixed variable and f_1 is a polynomial on fixed variables. After both of these u_i variables are fixed (via some extended Rule2), all variables are fixed, and it seems feasible to test the remaining polynomials with respect to the trusted set. Developing the theory and logic for PPE Circuits is an exciting future direction.

6 Conclusion

The ability to verify the well-formedness of a group of pairing elements (e.g., a private key) with respect to a set of trusted (e.g., public) parameters, by applying a set of pairing product equations, has numerous cryptographic applications. These include the design of basic and structure-preserving signature schemes, building oblivious transfer schemes from "blind" IBE, finding new verifiable random functions and keeping the IBE or ABE authority "accountable" to the user.

In this work, we provided original observations demonstrating that it is not always easy for a human to determine whether or not a public-private parameter pair can be verified using a set of PPEs. Many IBE schemes (e.g., [22, 29, 20, 45]) have PPE-testable parameters, but some IBE schemes in the literature, such as those based on dual-system encryption [47], provably do not.

To aid humans wishing to use PPE testability in their cryptographic design, we devised a set of rules for how to systematically search for a PPE testing set. We proved the correctness of this algorithm in Section 4.3 and provided an implementation of it, as AutoPPE, in Section 5. Tested on over two dozen schemes, the correctness and performance of the tool were solid. This allows researchers to move the discovery of PPE testing equations into the growing realm of cryptographic design tasks that can now be automated. This is one more important step in the larger goal of improving the speed and accuracy of the cryptographic design process via computer automation.

Acknowledgments

The authors are grateful to Brent Waters for valuable technical discussions and also to the anonymous reviewers of CCS 2019 for helpful feedback, especially for pointing out the connection to the automated analysis of structured-preserving signatures in [14].

References

[1] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. Cryptology ePrint Archive, Report 2012/285, 2012. `https://eprint.iacr.org/2012/285`.

[2] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Takeya Tango. Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In *Advances in Cryptology - CRYPTO*, pages 241–260. Springer, 2014.

[3] Masayuki Abe, Fumitaka Hoshino, and Miyako Ohkubo. Design in type-i, run in type-iii: Fast and scalable bilinear-type conversion using integer programming. In *Advances in Cryptology - CRYPTO*, pages 387–415. Springer, 2016.

[4] Joseph A. Akinyele, Gilles Barthe, Benjamin Grégoire, Benedikt Schmidt, and Pierre-Yves Strub. Certified synthesis of efficient batch verifiers. In *IEEE 27th Computer Security Foundations Symposium*, pages 153–165. IEEE Computer Society, 2014.

[5] Joseph A. Akinyele, Christina Garman, and Susan Hohenberger. Automating fast and secure translations from Type-I to Type-III pairing schemes. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1370–1381. ACM, 2015.

[6] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 399–410. ACM, 2013.

[7] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *the ACM Conference on Computer and Communications Security*, pages 474–487. ACM, 2012.

[8] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. *Journal of Computer Security*, 22(6):867–912, 2014.

[9] Miguel Ambrona, Gilles Barthe, Romain Gay, and Hoeteck Wee. Attribute-based encryption in the generic group model: Automated proofs and new constructions. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 647–664. ACM, 2017.

[10] Miguel Ambrona, Gilles Barthe, and Benedikt Schmidt. Automated unbounded analysis of cryptographic constructions in the generic group model. In *Advances in Cryptology - EUROCRYPT*, pages 822–851. Springer, 2016.

[11] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In *Advances in Cryptology - EUROCRYPT*, pages 689–718. Springer, 2015.

[12] Gilles Barthe, Francois Dupressoir, Benjamin Gregoire, Alley Stoughton, and Pierre-Yves Strub. Easycrypt: Computer-aided cryptographic proofs, 2018. `https://www.easycrypt.info/trac/`.

[13] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. In *Advances in Cryptology - CRYPTO*, pages 95–112. Springer, 2014.

[14] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt, and Mehdi Tibouchi. Strongly-optimal structure preserving signatures from type II pairings: Synthesis and lower bounds. In *Public-Key Cryptography - PKC 2015*, pages 355–376, 2015.

[15] Gilles Barthe, Xiong Fan, Joshua Gancher, Benjamin Grégoire, Charlie Jacomme, and Elaine Shi. Symbolic proofs for lattice-based cryptography. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 538–555. ACM, 2018.

[16] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 90–101. ACM, 2009.

[17] Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. Automated proofs of pairing-based cryptography. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1156–1168. ACM, 2015.

[18] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *2006 IEEE Symposium on Security and Privacy*, pages 140–154. IEEE Computer Society, 2006.

[19] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *Advances in Cryptology - EUROCRYPT*, pages 223–238. Springer, 2004.

[20] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459. Springer, 2004.

[21] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology - EUROCRYPT 2005*, pages 440–456, 2005.

[22] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO*, pages 213–229. Springer, 2001.

[23] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532. Springer, 2001.

[24] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *Advances in Cryptology - CRYPTO*, pages 290–307. Springer, 2006.

[25] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology - CRYPTO*, pages 56–72. Springer, 2004.

[26] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In *Public Key Cryptography - PKC*, pages 1–17. Springer, 2003.

[27] Francois Le Gall. Powers of tensors and fast matrix multiplication. *CoRR*, abs/1401.7714, 2014.

[28] Craig Gentry. Practical identity-based encryption without random oracles. In *Advances in Cryptology - EUROCRYPT*, pages 445–464. Springer, 2006.

[29] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *Advances in Cryptology - ASIACRYPT*, pages 548–566. Springer, 2002.

[30] Vipul Goyal. Reducing trust in the PKG in identity based cryptosystems. In *Advances in Cryptology - CRYPTO*, pages 430–447. Springer, 2007.

[31] Vipul Goyal, Steve Lu, Amit Sahai, and Brent Waters. Black-box accountable authority identity-based encryption. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security*, pages 427–436. ACM, 2008.

[32] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *Advances in Cryptology - ASIACRYPT*, pages 265–282. Springer, 2007.

[33] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, pages 415–432. Springer, 2008.

[34] Florian Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography*, pages 310–324. Springer, 2002.

[35] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 84–95. ACM, 2015.

[36] Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In *EUROCRYPT*, pages 656–672. Springer, 2010.

[37] Tibor Jager. Verifiable random functions from weaker assumptions. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC*, pages 121–143. Springer, 2015.

[38] Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In *Advances in Cryptology - EUROCRYPT 2011*, pages 547–567. Springer, 2011.

[39] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198. Springer, 2012.

[40] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *Advances in Cryptology - CRYPTO*, pages 597–612. Springer, 2002.

[41] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE 27th Computer Security Foundations Symposium*, pages 140–152. IEEE Computer Society, 2014.

[42] David Naccache. Secure and *Practical* identity-based encryption. *IACR Cryptology ePrint Archive*, 2005.

[43] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. In *Advances in Cryptology - ASIACRYPT*, pages 349–366. Springer, 2012.

[44] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 463–474. ACM, 2013.

[45] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127. Springer, 2005.

[46] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636. Springer, 2009.

[47] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636. Springer, 2009.

A Waters05 IBE Scheme

In this section, we recall the public parameters and the private keys from the Waters05 IBE scheme with Naccache's optimization [45, 42]. A part of the text has been taken verbatim from [42]. Let $\mathbb{G}_1$ be a group of prime order p, and g be a group generator for $\mathbb{G}_1$. Let e be an admissible bilinear map $e : \mathbb{G}_1 * \mathbb{G}_1 \to \mathbb{G}_T$. Identities will be represented as n dimensional vectors $v = (v_1, \ldots, v_n)$ where each v_i is an l-bit integer. The integers n and l are parameters unrelated to p, and $n' = n \cdot l$ is the output length of a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^{n'}$.

$\mathsf{Setup}(1^\lambda)$: Sample α, β, u and each element of an n-dimensional vector $U = (u_i)$ uniformly at random from $\mathbb{Z}_p$. Set $g_1 = g^\alpha$, $g_2 = g^\beta$, $z = g^u$ and $z_i = g^{u_i}$ for each $i \in [n]$. The public parameters pk are $g, g_1, g_2, z, \{z_i : i \in [n]\}$. The master secret key msk is g_2^α.

$\mathsf{KeyGen}(\mathsf{msk}, v)$: Let $v = (v_1, \ldots, v_n) \in (\{0,1\}^l)^n$ be an identity. Sample r be uniformly at random in $\mathbb{Z}_p$. The private key sk_v for identity v is constructed as

$$\mathsf{sk}_v = \left(g_2^\alpha \cdot \left(z \cdot \prod_{i=1}^{n} z_i^{v_i}\right)^r, g^r\right).$$

B More Case Study Examples

In this section, we present the PPEs output by the tool on few of the standard schemes.

B.1 Boneh-Boyen 04a (BB1) HIBE scheme

In this section, we recall the public parameters and the private keys of the BB1 HIBE [19] scheme.

$\mathsf{Setup}(1^\lambda, 1^\ell)$: To generate system parameters for an HIBE of maximum depth ℓ, select a random generator g in $\mathbb{G}_1$, and random $\alpha, x, h_1, h_2, \ldots, h_\ell \leftarrow \mathbb{Z}_p$, and set $g_1 = g^\alpha, g_2 = g^x, H_i = g^{h_i}$ for all $i \in [\ell]$. The public parameters and the master secret key are given by $\mathsf{pp} = (g, g_1, g_2, H_1, \ldots, H_\ell)$ and $\mathsf{msk} = g_2^\alpha$.

$\mathsf{KeyGen}(\mathsf{sk}_{\mathsf{id}|j-1}, \mathsf{id})$: To generate the private key $\mathsf{sk}_{\mathsf{id}}$ for an identity $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_j) \in \mathbb{Z}_p^j$ of depth $j \leq \ell$. Pick random $r_1, \ldots, r_j \leftarrow \mathbb{Z}_p$ and output

$$\mathsf{sk}_{\mathsf{id}} = \left(g_2^\alpha \cdot \prod_{k=1}^{j} (g_1^{id_k} \cdot H_k)^{r_k}, g^{r_1}, \ldots, g^{r_j}\right).$$

Note that the private key for id can be generated just given a private key for $\mathsf{id}|j-1 = (\mathsf{id}_1, \ldots, \mathsf{id}_{j-1}) \in \mathbb{Z}_p^{j-1}$ as required.

The input file for the BB1 HIBE scheme when the maximum depth $\ell = 3$ is presented in Figure 8. The output by the tool is described in Figure 9.

Input File for BB1 HIBE

```
(*BB1 HIBE scheme when number of levels is fixed to be 3*)
maps G1 * G1 ->GT.
fixed_vars [alpha, x, h1, h2, h3].
unfixed_vars [r1, r2, r3].
trusted_polys [F1 = alpha, F2=x, F3=h1, F4=h2, F5=h3] in G1.
Zp_vars [id1, id2, id3].
untrusted_polys [F6=x*alpha + (alpha*id1+h1)*r1 + (alpha*id2+h2)*r2 + (alpha*id3+h3)*r3, F7=r1, F8=r2, F9=r3] in
G1.
```

Figure 8: Input file for BB1 HIBE scheme.

Output of the tool for BB1 HIBE

```
F0 = 1 in G1       F0 = 1 in GT       F1 = alpha in G1       F2 = x in G1
F3 = h1 in G1      F4 = h2 in G1      F5 = h3 in G1
F6 = alpha*x + h1*r1 + h2*r2 + h3*r3 + alpha*id1*r1 + alpha*id2*r2 + alpha*id3*r3 in G1
F7 = r1 in G1      F8 = r2 in G1      F9 = r3 in G1
F10 = id1 in G1    F11 = id2 in G1    F12 = id3 in G1
F13 = id1 in GT    F14 = id2 in GT    F15 = id3 in GT
F16 = alpha*id1 in G1    F17 = alpha*id2 in G1    F18 = alpha*id3 in G1
F19 = id1*x in G1    F20 = id2*x in G1    F21 = id3*x in G1
F22 = h1*id1 in G1    F23 = h1*id2 in G1    F24 = h1*id3 in G1
F25 = h2*id1 in G1    F26 = h2*id2 in G1    F27 = h2*id3 in G1
F28 = h3*id1 in G1    F29 = h3*id2 in G1    F30 = h3*id3 in G1

Processing untrusted polynomial F7 = r1 by rule2
F7 moved to trusted set and r1 moved to fixed set by rule 2

Processing untrusted polynomial F8 = r2 by rule2
F8 moved to trusted set and r2 moved to fixed set by rule 2

Processing untrusted polynomial F6 = alpha*x + h1*r1 + h2*r2 + h3*r3 + alpha*id1*r1 + alpha*id2*r2 + alpha*id3*r3
by rule2
Rule not applied

Processing untrusted polynomial F9 = r3 by rule2
F9 moved to trusted set and r3 moved to fixed set by rule 2

Processing untrusted polynomial F6 by rule1
Naive PPE e(F6,F0) = e(F1,F2) * e(F3,F7) * e(F4,F8) * e(F5,F9) * e(F7,F16) * e(F8,F17) * e(F9,F18)
Optimized PPE e(F6,F0) = e(F1,F2)*e(F3*F16,F7)*e(F4*F17,F8)*e(F5*F18,F9)
F6 moved to trusted set by rule 1

PPEs : e(F6,F0) = e(F1,F2)*e(F3*F16,F7)*e(F4*F17,F8)*e(F5*F18,F9)
Ouptut : PPE Testable :)
```

Figure 9: Output of the tool for BB1 HIBE scheme.

B.2 Camenisch-Lysyanskaya Signature Scheme

In this section, we recall the Camenisch-Lysyanskaya Signature Scheme B [25] signature scheme adapted to type-III pairings. We note that the original scheme described in the paper is in type-I setting. The setup and signing procedures of the scheme proceeds as follows.

$\mathsf{Setup}(1^\lambda)$: Select a random generator g_1 in group $\mathbb{G}_1$ and g_2 in group $\mathbb{G}_2$. Sample random values $x, y, z \leftarrow \mathbb{Z}_p$. Set $X = g_1^x, Y = g_1^y, Z = g_1^z$. Set verification key and secret key as $\mathsf{vk} = (g_1, g_2, X, Y, Z)$ and $\mathsf{sk} = (x, y, z)$.

$\mathsf{Sign}(\mathsf{vk}, \mathsf{sk}, \mathsf{msg})$: Parse input message as $\mathsf{msg} = (m, r) \in \mathbb{Z}_p^2$, and secret key as $\mathsf{sk} = (x, y, z)$ and verification key as $\mathsf{vk} = (g_1, g_2, X, Y, Z)$. Sample a random $a \leftarrow \mathbb{Z}_p$. Set $A = g_2^{az}$, $b = g_2^{ay}$, $B = g_2^{azy}$, $c = g_2^{ax+axym+azxyr}$. Output signature $\sigma = (g_2^a, A, b, B, c)$.

The input file for CL04 Signature Scheme B [25] is presented in Figure 10. The output of the tool is presented in Figure 11.

B.3 Waters09 IBE Scheme

In this section, we recall Waters09 HIBE scheme. The setup and key generation algorithms of the scheme proceeds as follows.

Input File for CL04 Signature

```
maps G1 * G2 ->GT.
fixed_vars [x, y, z].
unfixed_vars [a].
Zp_vars [m, r]. (*message*)
trusted_polys [F1 = x, F2 = y, F3 = z] in G1. (*verification key*)
untrusted_polys [F4 = a*z, F5 = a*y, F6 = a*z*y, F7 = a*(x + x*y*m) + a*z*x*y*r, F8 = a] in G2. (*signature*)
```

Figure 10: Input file for CL04 signature scheme B.

Output of the tool for CL04 Signature Scheme B

```
F0 = 1 in G1      F0 = 1 in G2      F0 = 1 in GT      F1 = x in G1      F2 = y in G1      F3 = z in G1
F4 = a*z in G2      F5 = a*y in G2      F6 = a*y*z in G2      F7 = a*x + a*m*x*y + a*r*x*y*z in G2
F8 = a in G2      F9 = m in G1      F10 = r in G1      F11 = m in G2      F12 = r in G2
F13 = m in GT      F14 = r in GT      F15 = m*x in G1      F16 = r*x in G1      F17 = m*y in G1
F18 = r*y in G1      F19 = m*z in G1      F20 = r*z in G1

Processing untrusted polynomial F4 = a*z by rule2      Rule not applied
Processing untrusted polynomial F5 = a*y by rule2      Rule not applied
Processing untrusted polynomial F6 = a*y*z by rule2      Rule not applied
Processing untrusted polynomial F7 = a*x + a*m*x*y + a*r*x*y*z by rule2      Rule not applied

Processing untrusted polynomial F8 = a by rule2
F8 moved to trusted set and a moved to fixed set by rule 2

Processing untrusted polynomial F4 by rule1
Naive PPE e(F4,F0) = e(F3,F8)
Optimized PPE e(F4,F0) = e(F3,F8)
F4 moved to trusted set by rule 1

Processing untrusted polynomial F5 by rule1
Naive PPE e(F5,F0) = e(F2,F8)
Optimized PPE e(F5,F0) = e(F2,F8)
F5 moved to trusted set by rule 1

Processing untrusted polynomial F6 by rule1
Naive PPE e(F6,F0) = e(F2,F4)
Optimized PPE e(F6,F0) = e(F2,F4)
F6 moved to trusted set by rule 1

Processing untrusted polynomial F7 by rule1
Naive PPE e(F7,F0) = e(F16,F6) * e(F15,F5) * e(F1,F8)
Optimized PPE e(F7,F0) = e(F16,F6)*e(F15,F5)*e(F1,F8)
F7 moved to trusted set by rule 1

PPEs : e(F7,F0) = e(F16,F6)*e(F15,F5)*e(F1,F8), e(F6,F0) = e(F2,F4), e(F5,F0) = e(F2,F8), e(F4,F0) = e(F3,F8)
Ouptut : PPE Testable :)
```

Figure 11: Output of the tool for CL04 Signtature Scheme B.

$\mathsf{Setup}(1^\lambda)$: Sample a random generator $g \leftarrow \mathbb{G}_1$, and then sample random elements $v, v_1, v_2, w, u, h, a_1, a_2, b, \alpha \leftarrow \mathbb{Z}_p$. Set $V = g^v, V_1 = g^{v_1}, V_2 = g^{v_2}, W = g^w, U = g^u, H = g^h, \mathrm{T}_1 = g^{v+v_1 a_1}, \mathrm{T}_2 = g^{v+v_2 a_2}$. It then public parameters

$$\mathsf{pp} = (g^b, g^{a_1}, g^{a_2}, g^{b\cdot a_1}, g^{b\cdot a_2}, \mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_1^b, \mathrm{T}_2^b, W, U, H, e(g,g)^{\alpha\cdot a_1\cdot b}).$$

It sets master secret key $\mathsf{msk} = (g, g^{\alpha}, g^{\alpha \cdot a_1}, V, V_1, V_2)$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$: Sample random elements $r_1, r_2, z_1, z_2, \mathsf{tag}_k \leftarrow \mathbb{Z}_p$. Let $r = r_1 + r_2$. Output secret key

$$\mathsf{sk}_{\mathsf{id}} = \Big(g^{\alpha \cdot a_1} \cdot V^r, g^{-\alpha} V_1^r g^{z_1}, (g^b)^{-z_1}, V_2^r g^{z_2}, (g^b)^{-z_2}, g^{r_2 b}, g^{r_1}, (U^{\mathsf{id}} W^{\mathsf{tag}_k} H)^{r_1}, \mathsf{tag}_k\Big).$$

The input file for Waters09 IBE scheme [46] is presented in Figure 12. The output by the tool is presented in Figure 14. Note that, this scheme is provably not PPE Testable, and hence our tool outputs unknown.

Input File for Waters09 IBE

```
maps G1 * G1 ->GT.
fixed_vars [a1, a2, b, alpha, v, v1, v2, w, u, h].
unfixed_vars [r1, r2, z1, z2].
Zp_vars [id, tag].
trusted_polys [F1 = b, F2 = a1, F3 = a2, F4 = b*a1, F5 = b*a2, F6 = v+v1*a1, F7 = v+v2*a2, F8 = b*(v+v1*a1), F9 =
b*(v+v2*a2), F10 = w, F11 = u, F12 = h] in G1.
trusted_polys [F13 = alpha*a1*b] in GT.
untrusted_polys [F14 = alpha*a1+v*(r1+r2), F15 = -alpha+v1*(r1+r2)+z1, F16 = -b*z1, F17 = v2*(r1+r2) + z2, F18 =
-b*z2, F19 = r2*b, F20 = r1, F21 = (u*id+w*tag+h)*r1 ] in G1.
```

Figure 12: Input file for Waters09 IBE scheme.

Input File for RW13 CP-ABE

```
(*Rouselakis Waters CP-ABE construction with k = 4*)
maps G1 * G1 ->GT.
fixed_vars [u, h, w, v, alpha].
unfixed_vars [r, r1, r2, r3, r4].
(*public key*)
trusted_polys [F1 = u, F2 = h, F3 = w, F4 = v] in G1.
trusted_polys [F5 = alpha] in GT.
Zp_vars [a1, a2, a3, a4]. (*attributes*)
(*Secret key*)
untrusted_polys [F6 = alpha + w*r, F7 = r] in G1.
untrusted_polys [F8 = (u*a1 + h)*r1 - v*r, F9 = r1] in G1.
untrusted_polys [F9 = (u*a2 + h)*r2 - v*r, F10 = r2] in G1.
untrusted_polys [F10 = (u*a3 + h)*r3 - v*r, F11 = r3] in G1.
untrusted_polys [F12 = (u*a4 + h)*r4 - v*r, F13 = r4] in G1.
```

Figure 13: Input file for RW13 CP-ABE scheme.

B.4 Rouselakis-Waters CP-ABE Scheme

In this section, we recall Rouselakis-Waters CP-ABE scheme [44]. The setup and key generation algorithms of the scheme proceeds as follows.

$\mathsf{Setup}(1^{\lambda})$: The algorithm picks a random generator $g \leftarrow \mathbb{G}_1$, samples $u, h, w, v, \alpha \leftarrow \mathbb{Z}_p$ and sets $U = g^u, H = g^h, W = g^w, V = g^v$. It outputs public parameters $\mathsf{pp} = (g, U, H, W, V, e(g,g)^{\alpha})$ and $\mathsf{msk} = \alpha$.

$\mathsf{KeyGen}(\mathsf{msk}, S = \{a_1, a_2, \ldots, a_k\} \subseteq \mathbb{Z}_p)$: Initially, the key generation algorithm picks $k+1$ random exponents $r, r_1, r_2, \ldots, r_k \leftarrow \mathbb{Z}_p$. Then it computes $K_0 = g^{\alpha} \cdot W^r$, $K_1 = g^r$, and for every $i \in [k]$ it computes $K_{i,2} = g^{r_i}$ and $K_{i,3} = (U^{a_i} H)^{r_i} V^{-r}$. The secret key output is $\mathsf{sk} = (S, K_0, K_1, \{K_{i,2}, K_{i,3}\} \forall i \in [k])$.

The input file for Rouselakis-Waters CP-ABE scheme [44] when the number of attributes is fixed to be 4 is presented in Figure 13. The output by the tool is presented in Figure 15.

Output of the tool for Waters09 IBE

```
F0 = 1 in G1      F0 = 1 in GT      F1 = b in G1      F2 = a1 in G1      F3 = a2 in G1
F4 = a1*b in G1      F5 = a2*b in G1      F6 = v + a1*v1 in G1      F7 = v + a2*v2 in G1
F8 = b*v + a1*b*v1 in G1      F9 = b*v + a2*b*v2 in G1
F10 = w in G1      F11 = u in G1      F12 = h in G1      F13 = a1*alpha*b in GT
F14 = a1*alpha + r1*v + r2*v in G1      F15 = -1*alpha + z1 + r1*v1 + r2*v1 in G1
F16 = -1*b*z1 in G1      F17 = z2 + r1*v2 + r2*v2 in G1      F18 = -1*b*z2 in G1
F19 = b*r2 in G1      F20 = r1 in G1      F21 = h*r1 + id*r1*u + r1*tag*w in G1
F22 = id in G1      F23 = tag in G1      F24 = id in GT      F25 = tag in GT
F26 = b*id in G1      F27 = b*tag in G1      F28 = a1*id in G1      F29 = a1*tag in G1
F30 = a2*id in G1      F31 = a2*tag in G1      F32 = a1*b*id in G1
F33 = a1*b*tag in G1      F34 = a2*b*id in G1      F35 = a2*b*tag in G1
F36 = id*v + a1*id*v1 in G1      F37 = tag*v + a1*tag*v1 in G1
F38 = id*v + a2*id*v2 in G1      F39 = tag*v + a2*tag*v2 in G1
F40 = b*id*v + a1*b*id*v1 in G1      F41 = b*tag*v + a1*b*tag*v1 in G1
F42 = b*id*v + a2*b*id*v2 in G1      F43 = b*tag*v + a2*b*tag*v2 in G1
F44 = id*w in G1      F45 = tag*w in G1      F46 = id*u in G1      F47 = tag*u in G1
F48 = h*id in G1      F49 = h*tag in G1      F50 = a1*alpha*b*id in GT      F51 = a1*alpha*b*tag in GT

Processing untrusted polynomial F16 = -1*b*z1 by rule2      Rule not applied
Processing untrusted polynomial F18 = -1*b*z2 by rule2      Rule not applied
Processing untrusted polynomial F19 = b*r2 by rule2      Rule not applied

Processing untrusted polynomial F20 = r1 by rule2
F20 moved to trusted set and r1 moved to fixed set by rule 2

Processing untrusted polynomial F14 = a1*alpha + r1*v + r2*v by rule2      Rule not applied
Processing untrusted polynomial F16 = -1*b*z1 by rule2      Rule not applied
Processing untrusted polynomial F18 = -1*b*z2 by rule2      Rule not applied
Processing untrusted polynomial F19 = b*r2 by rule2      Rule not applied

Processing untrusted polynomial F21 by rule1
Naive PPE e(F21,F0) = e(F12,F20) * e(F20,F45) * e(F20,F46)
Optimized PPE e(F21,F0) = e(F12*F45*F46,F20)
F21 moved to trusted set by rule 1

Processing untrusted polynomial F14 = a1*alpha + r1*v + r2*v by rule2      Rule not applied
Processing untrusted polynomial F16 = -1*b*z1 by rule2      Rule not applied
Processing untrusted polynomial F18 = -1*b*z2 by rule2      Rule not applied
Processing untrusted polynomial F19 = b*r2 by rule2      Rule not applied

Execution time : 2.644071s
Untrusted set : F14, F15, F16, F17, F18, F19
PPEs : e(F21,F0) = e(F12*F45*F46,F20)
Output : Unknown
```

Figure 14: Output of the tool for Waters09 IBE scheme.

Output of the tool for RW13 CP-ABE

```
F0 = 1 in G1      F0 = 1 in GT      F1 = u in G1      F2 = h in G1      F3 = w in G1      F4 = v in G1
F5 = alpha in GT      F6 = alpha + r*w in G1      F7 = r in G1
F8 = h*r1 - r*v + a1*r1*u in G1      F9 = r1 in G1      F10 = h*r2 - r*v + a2*r2*u in G1      F11 = r2 in G1
F12 = h*r3 - r*v + a3*r3*u in G1      F13 = r3 in G1      F14 = h*r4 - r*v + a4*r4*u in G1      F15 = r4 in G1
F16 = a1 in G1      F17 = a2 in G1      F18 = a3 in G1      F19 = a4 in G1
F20 = a1 in GT      F21 = a2 in GT      F22 = a3 in GT      F23 = a4 in GT
F24 = a1*u in G1      F25 = a2*u in G1      F26 = a3*u in G1      F27 = a4*u in G1
F28 = a1*h in G1      F29 = a2*h in G1      F30 = a3*h in G1      F31 = a4*h in G1
F32 = a1*w in G1      F33 = a2*w in G1      F34 = a3*w in G1      F35 = a4*w in G1
F36 = a1*v in G1      F37 = a2*v in G1      F38 = a3*v in G1      F39 = a4*v in G1
F40 = a1*alpha in GT      F41 = a2*alpha in GT      F42 = a3*alpha in GT      F43 = a4*alpha in GT

Processing untrusted polynomial F6 = alpha + r*w by rule2.      Rule not applied
Processing untrusted polynomial F7 = r by rule2
F7 moved to trusted set and r moved to fixed set by rule 2
Processing untrusted polynomial F8 = h*r1 - r*v + a1*r1*u by rule2      Rule not applied
Processing untrusted polynomial F9 = r1 by rule2
F9 moved to trusted set and r1 moved to fixed set by rule 2
Processing untrusted polynomial F10 = h*r2 - r*v + a2*r2*u by rule2      Rule not applied
Processing untrusted polynomial F11 = r2 by rule2
F11 moved to trusted set and r2 moved to fixed set by rule 2
Processing untrusted polynomial F12 = h*r3 - r*v + a3*r3*u by rule2      Rule not applied
Processing untrusted polynomial F13 = r3 by rule2
F13 moved to trusted set and r3 moved to fixed set by rule 2
Processing untrusted polynomial F14 = h*r4 - r*v + a4*r4*u by rule2      Rule not applied
Processing untrusted polynomial F15 = r4 by rule2
F15 moved to trusted set and r4 moved to fixed set by rule 2

Processing untrusted polynomial F6 by rule1
Naive PPE e(F6,F0) = F5 * e(F3,F7)
Optimized PPE e(F6,F0) = F5*e(F3,F7)
F6 moved to trusted set by rule 1

Processing untrusted polynomial F8 by rule1
Naive PPE e(F8,F0) = e(F2,F9) * (e(F4,F7))^-1 * e(F9,F24)
Optimized PPE e(F8,F0) = (e(F4,F7))^-1*e(F2*F24,F9)
F8 moved to trusted set by rule 1

Processing untrusted polynomial F10 by rule1
Naive PPE e(F10,F0) = e(F2,F11) * (e(F4,F7))^-1 * e(F11,F25)
Optimized PPE e(F10,F0) = (e(F4,F7))^-1*e(F2*F25,F11)
F10 moved to trusted set by rule 1

Processing untrusted polynomial F12 by rule1
Naive PPE e(F12,F0) = e(F2,F13) * (e(F4,F7))^-1 * e(F13,F26)
Optimized PPE e(F12,F0) = (e(F4,F7))^-1*e(F2*F26,F13)
F12 moved to trusted set by rule 1

Processing untrusted polynomial F14 by rule1
Naive PPE e(F14,F0) = e(F2,F15) * (e(F4,F7))^-1 * e(F15,F27)
Optimized PPE e(F14,F0) = (e(F4,F7))^-1*e(F2*F27,F15)
F14 moved to trusted set by rule 1

Execution time : 12.837395s
PPEs : e(F14,F0) = (e(F4,F7))^-1*e(F2*F27,F15), e(F12,F0) = (e(F4,F7))^-1*e(F2*F26,F13), e(F10,F0) = (e(F4,F7))^-
1*e(F2*F25,F11), e(F8,F0) = (e(F4,F7))^-1*e(F2*F24,F9), e(F6,F0) = F5*e(F3,F7)
Ouptut : PPE Testable :)
```

Figure 15: Output of the tool for RW13 CP-ABE scheme.