PPE Circuits for Rational Polynomials

Susan Hohenberger
Johns Hopkins University
Baltimore, MD, USA
susan@cs.jhu.edu

Satyanarayana Vusirikala
University of Texas at Austin
Austin, TX, USA
satya@cs.utexas.edu

ABSTRACT

Pairings are a powerful algebraic setting for realizing cryptographic functionalities. One challenge for cryptographers who design pairing systems is that the complexity of many systems in terms of the number of group elements and equations to verify has been steadily increasing over the past decade and is approaching the point of being unwieldy. To combat this challenge, multiple independent works have utilized computers to help with the system design. One common design task that researchers seek to automate is summarized as follows: given a description of a set of trusted elements T (e.g., a public key) and a set of untrusted elements U (e.g., a signature), automatically generate an algorithm that verifies U with respect to T using the pairing and group operations. To date, none of the prior automation works for this task have support for solutions with rational polynomials in the exponents despite many pairing constructions employing them (e.g., Boneh-Boyen signatures, Gentry's IBE, Dodis-Yampolskiy VRF).

We demonstrate how to support this essential class of pairing systems for automated exploration. Specifically, we present a solution for automatically generating a verification algorithm with novel support for rational polynomials. The class of verification algorithms we consider in this work is called PPE Circuits (introduced in [HVW20]). Intuitively, a PPE Circuit is a circuit supporting pairing and group operations, which can test whether a set of elements U verifies with respect to a set of elements T. We provide a formalization of the problem, an algorithm for searching for a PPE Circuit supporting rational polynomials, a software implementation, and a detailed performance evaluation. Our implementation was tested on over three dozen schemes, including over ten test cases that our tool can handle, but prior tools could not. For all test cases where a PPE Circuit exists, the tool produced a solution in three minutes or less.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

Automated Design; Provable Security; Pairing-based Cryptography

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484562

ACM Reference Format:
Susan Hohenberger and Satyanarayana Vusirikala. 2021. PPE Circuits for Rational Polynomials. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3460120.3484562

1 INTRODUCTION

Computer automation has the potential to revolutionize the cryptographic design process, from discovering novel cryptographic functionalities to verifying their security. Computers can operate faster, with higher accuracy, and at a lower cost than the primarily manual process in place today. The key technical challenge is to devise (provably correct) algorithms that capture the human mind's incredible creativity in searching for a scheme with the desired functionality or in devising an approach for reducing the security of a cryptosystem to the hardness of a well-studied math problem.

Over the past decade, the field of cryptographic computer automation has exploded with promising results. There are software tools for building novel cryptographic algorithms [9, 10, 14, 20, 45–47, 51], translating schemes from one algebraic setting to another [3, 5, 7, 8, 55], analyzing the security of cryptographic assumptions [15, 19], strengthening the security of signatures [8] and automating proof generation and/or verification [18, 21–24]. Excitingly, these tools were employed to verify the security of protocols in Amazon Web Services Key Management Service [11], the cryptographic hash standard SHA-3 [13], key exchange protocols [17, 34], multiparty computation protocols [44], commitment schemes [52], software stacks [12], protocols in the Universal Composability framework [34] and even algorithms designed by other automated tools [6]. See Barbosa et al. [16] for a recent survey on cryptographic automation.

The goal of this work is to continue this momentum by presenting a novel tool for automating cryptographic design. Specifically, we focus on the pairing algebraic setting and put forth a tool that given a description of a set of trusted elements T (e.g., public parameters) and a set of untrusted elements U (e.g., an IBE private key) can automatically generate an algorithm that verifies U with respect to T using the pairing and group operations. What distinguishes our tool from prior works is that it supports schemes with rational polynomials (e.g., schemes with elements of the form $g^{1/a}$, $g^{b/(a+c)}$, etc.). This includes schemes such as Boneh-Boyen signatures [27] and identity-based encryption (IBE) [25], Gentry's IBE [37], the Dodis-Yampolskiy verifiable random function [36], the Le-Gabillon multisignatures [49], and more, which prior tools did not handle. Thus, we solve one of the major open problems for pairing-based automation [47]. We now describe our goals and contributions.

Pairing Product Equations (PPEs) and PPE Circuits. We focus on the pairing algebraic setting, which is known for its high speed, small bandwidth, and novel functionalities. The setting consists of groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p, and a *pairing* function which is an efficient, non-degenerate 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 $e(g^a, h^b) = e(g, h)^{ab}$. In some cases, $\mathbb{G}_1$ can be the same as $\mathbb{G}_2$. See Appendix A.1 for a formal treatment. This function is often leveraged in a cryptographic system to *verify* some elements with respect to others. E.g., it might be used to verify a signature using the public key.

Discovering correct verification algorithms can be extremely challenging. For instance, consider these public parameters for Waters Dual System Encryption [57]: $(g, w, u, h, \tau_1 = v v_1^{a_1}, \tau_2 = v v_2^{a_2}, \tau_1^b, \tau_2^b, g^b, g^{a_1}, g^{a_2}, g^{b a_1}, g^{b a_2}, e(g,g)^{\alpha a_1 b})$. Is it obvious how to use them with the pairing function to verify a private key of the form $(g^{\alpha a_1} v^{r_1+r_2}, g^{-\alpha} v_1^{r_1+r_2} g^{z_1}, g^{-b z_1}, v_2^{r_1+r_2} g^{z_2}, g^{-b z_2}, g^{r_2 b}, g^{r_1}, (u^I w^t h)^{r_1}))$, where I and t are public? No.[1]

To make searching for verification algorithms easier, multiple works [20, 46, 47] have employed computers to hunt for them and we build on this line of work. We begin our technical discussion by formalizing the concept of a "pairing verification algorithm" as a Pairing Product Equation (PPE) Circuit.

Following [43], a *pairing product equation* (PPE) over variables $Z, \{X_i\}_{i=1}^m, \{Y_i\}_{i=1}^n$ is an equation of the form

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

where $A_i, X_i \in \mathbb{G}_1, B_i, Y_i \in \mathbb{G}_2, Z \in \mathbb{G}_T, \gamma_{ij} \in \mathbb{Z}_p$. Following [47], a *PPE Circuit* is a circuit whose gates are AND, OR, NOT or PPEs; that is, a circuit that can evaluate PPEs together with other basic logic. (We define this formally in Section 2.)

Earlier Automated Discovery of PPEs and PPE Circuits. Barthe, Fagerholm, Fiore, Scedrov, Schmidt, and Tibouchi [20] presented an automated tool for designing optimal structure-preserving signatures in the pairing setting in 2015. The tool considers increasingly larger candidates for the public key and signature structure, testing to see if there is a conjunction of PPEs that can verify the signature with respect to the public key. In 2019, Hohenberger and Vusirikala [46] generalized this approach with a tool that takes in a description of any set of trusted pairing elements T and untrusted pairing elements U and searches for a conjunction of PPEs that can verify U with respect to T. In 2020, Hohenberger, Vusirikala, and Waters [47] formalized the concept of a PPE Circuit and presented the AutoCircuitPPE tool[2], which takes as input the description of the sets T and U and searches for a PPE Circuit that can verify U with respect to T. They demonstrated that increasing the power of the verification algorithm to arbitrary logic over PPEs enabled the discovery of several novel verification algorithms including the first algorithm to verify the Boyen-Waters anonymous IBE [31] private keys with respect to the public key using 27 PPEs and 124 boolean gates.

[1] Indeed, no efficient verification algorithm even exists assuming that the Waters Dual System Encryption scheme is secure [46], even though prior works [1] were interested in finding such an algorithm.

[2] https://github.com/JHUISI/auto-tools

Limitation of Prior Works. In prior works [20, 46, 47], all elements of T and U had to be of the form $g^{f(\mathbf{u})}$ for some multivariate polynomial f over variables $\mathbf{u} = \{u_1, \ldots, u_k\}$. These tools did not allow elements with rational polynomials, such as $g^{f(\mathbf{u})/h(\mathbf{u})}$, to be in T or U, because for various settings of the variables $\mathbf{u}$, the denominator $h(\mathbf{u})$ may evaluate to zero making the element $g^{f(\mathbf{u})/h(\mathbf{u})}$ undefined and the prior algorithms did not address how to handle/prevent these undefined elements. However, many pairing-based cryptosystems use rational polynomials (e.g., [2, 25, 27, 36, 37, 49]). Thus, finding a way to support rational polynomials in automated cryptographic design was viewed as an important open problem [47].

1.1 Summary of Our Results

This work presents a formalization (Section 2), an algorithm (Section 3) and software (Section 4) that takes as input a description of trusted pairing elements T and untrusted pairing elements U, where elements of either set can be of the form $g^{f(\mathbf{u})/h(\mathbf{u})}$ for multivariate polynomials f, h over variables $\mathbf{u} = \{u_1, \ldots, u_k\}$, and outputs either a PPE Circuit that verifies U with respect to T or the special symbol unknown. The property we require (and prove) of our automator is that if it outputs a circuit, the circuit correctly verifies *any* U given a valid T. However, outputting unknown is not a guarantee that no circuit exists. Our algorithm (see Figure 7) is a superset of prior logic [20, 46, 47], finding all schemes they could, and with the addition of new logic many more. We name the implemented tool AutoRationalPPE.

We tested AutoRationalPPE on over thirty-five test cases, including over ten cases that our tool could handle, but prior tools could not. These newly successful test cases include the Boneh-Boyen signatures [27], Boneh-Boyen IBE [25], Gentry's IBE [37], the Dodis-Yampolskiy verifiable random function [36], the Le-Gabillon multisignatures [49] and more. We included several custom test cases to test schemes with over 100 elements, invoke different subroutines, etc. See Table 1.

For test cases where a PPE Circuit exists, AutoRationalPPE output a solution in 3 minutes or less. Furthermore, for 90% of the cases, it took under one minute. For schemes with solutions, the test case that took the longest was the Boneh-Boyen Hierarchical Identity Based Encryption [25] with 160 levels, which took almost three minutes. We designed Custom Testcase 6 specifically to challenge the tool with over 100 elements. For this test case, T contains g^a and U contains $g^{a^2}, g^{a^3}, \ldots, g^{a^{99}}, g^{1/a^{100}}, g^{1/a}$. The tool took under 4 seconds to output a solution with 102 PPEs and 103 boolean gates.

We include two test cases that provably do not have PPE Circuits (including the impossibility of verifying the private keys of [57] with respect to its public parameters, since this system has semi-functional keys that are not in the private-key space but cannot be distinguished from private keys). For both of these cases, AutoRationalPPE correctly aborts and outputs unknown. It took 5.5 minutes to output unknown on [57], which was its longest-running time in our tests.

Overall, we believe the tool is easy to understand and quick enough for practical use. The source code of AutoRationalPPE is publicly available at https://github.com/JHUISI/auto-tools.

1.2 Technical Overview

As stated in Section 1.1, the main algorithm of AutoRationalPPE (in Figure 7) takes as input a description of sets T and U and outputs either a PPE Circuit or the symbol unknown.

This algorithm is recursive. The base case is when there are no untrusted elements ($U = \emptyset$) and in this case, a trivial PPE Circuit that outputs 1 on all inputs is the output. When $U \neq \emptyset$, the algorithm tries to "move" a single element $F \in U$ to T by seeing if there is a test (that can be encoded as a circuit C) that validates F. If it cannot find any $F \in U$ that it can "move", then it aborts and outputs unknown. If it can "'move" an F, then it recurses on the (smaller) subproblem where $T' = T \cup \{F\}$ and $U' = U/\{F\}$. Suppose circuit C' is the output of the call on T' and U'. The PPE Circuit output is a combination[3] of C and C'.

This is our solution in a nutshell. The technical core of this algorithm is in (1) how an element can be "moved" from U to T and (2) how to build the final PPE Circuit as a combination of subproblem circuits. We tackle the first challenge by applying one of the four rules outlined below.

For all descriptions below, let $F = g^{f(\mathbf{u})/h(\mathbf{u})}$. Let InTrusted be the set of variables in $\mathbf{u}$ that appear in any element in T. So, if $T = \{g^a, g^b\}$ and $U = \{g^{ab}, g^{ac}, g^{ac/(a+b)}, g^{1/(d+a)}\}$, then $\mathbf{u} = \{a, b, c, d\}$ and InTrusted $= \{a, b\}$. Let Space(T) be the set of elements that can be computed using elements of T.

Rules 1 and 2 are for when F contains only variables in InTrusted. Rules 3a and 3b are for when F contains one or more variables not in InTrusted; these are necessary to handle the fresh randomness used in private keys and signatures, etc.

Rule 1 (Figure 2): This is a simple rule. It moves $F \in U$ to T if (1) all variables in F are in InTrusted, (2) $A = e(F, g)$ for some $A \in$ Space(T) and (3) $h(\mathbf{u}) \neq 0$. Here F can *only* be paired with the *generator* g. In our example, we can check $e(g^a, g^b) = e(F, g)$ to test if F is g^{ab}.

Rule 2 (Figure 4): This is a generalization of Rule 1 that allows F to be paired with an element $F' \in$ Space(T), instead of only the generator g. It is less efficient than Rule 1, because it requires both our recursive algorithm and the PPE Circuit it builds to condition on whether or not $F' = g^0$. (Because the equation $e(g, g)^0 = e(F, F')$ is trivially satisfied for all F when $F' = g^0$ and thus cannot be used to validate F in this case.)

In the first branch, we check if (1) $F' \neq g^0$, (2) all variables in F are in InTrusted, (3) there exists an $A \in$ Space(T) such that $A = e(F, F')$ and (4) $h(\mathbf{u}) \neq 0$. If all conditions are met, it moves F to T', recurses on that smaller problem and adds this logic to the PPE Circuit with a validation for F and a check that $F' \neq g^0$. For instance, $e(g^{ac}, g) = e(F, g^{a+b})$ can be used to check that F is $g^{ac/(a+b)}$ when $a + b \neq 0$.

In the second branch where $F' = g^0$, the algorithm recurses on the (possibly) reduced sets T', U' where zero is substituted for the exponent of F'. It also adds any subcircuit produced for this problem to the PPE circuit together with a check that $F' = g^0$. In our example, where $T = \{g^a, g^b\}$ and $U = \{g^{ab}, g^{ac}, g^{ac/(a+b)}, g^{1/(d+a)}\}$, if we substitute $a + b = 0$ into these sets, element $g^{ac/(a+b)}$ becomes undefined. This branch is thus not allowed and returns a circuit that always rejects.

The logic from these two branches will be connected with an OR gate.

Rule 3a (Figure 5): Rule 3a handles $F = g^{f(\mathbf{u})/h(\mathbf{u})}$ in U where the numerator $f(\mathbf{u})$ contains a variable not in InTrusted, but all variables of the denominator $h(\mathbf{u})$ are in InTrusted. Here we look at numerators $f(\mathbf{u}) = h' u_j^d + h''$, where (1) u_j is not in InTrusted, 2) h' and $h(\mathbf{u})$ contain only variables in InTrusted, (3) h'' does not contain u_j, (4) d is relatively prime to $p - 1$ and (5) $h(\mathbf{u}) \neq 0$. This rule has a larger potential branching degree than Rule 2, because it must additionally branch on whether $h' \neq 0$ since it cannot use F to validate u_j if u_j is zeroed out by h' in F.

In our example, Rule 3a would identify $F = g^{ac}$ as a candidate to move to trusted as c is not in InTrusted. It must condition on whether $a = 0$. In the branch where $a \neq 0$, it would move g^{ac} to trusted and set InTrusted $= \{a, b, c\}$. In the branch where $a = 0$, it performs this substitution and recurses on the subproblem $T' = \{g^0, g^b\}$ and $U' = \{g^0, g^0, g^{c/b}, g^{1/d}\}$.

Rule 3b (Figure 6): Rule 3b is the same as Rule 3a, except now all variables of the numerator must be in InTrusted and the denominator contains a variable not in InTrusted. Using our example, we consider the variable d in element $F = g^{1/(d+a)}$. Since there is no coefficient for variable d, it will not need to branch, and the rule will move F to T, and the main algorithm will recurse on this smaller problem.

Tracking Zeros. Critically to our support for rational polynomials, our main algorithm and rules also contain logic to ensure that at no point is a denominator of any element zero or reduced with a polynomial that evaluates to zero, because without this check, we could erroneously lose track of when an element becomes undefined. For instance, suppose the polynomial for an exponent of an element is $a(a + c)/(b(a + c))$. We cannot reduce this to a/b unless we first check that $a + c \neq 0$.

Putting It All Together. Each time we apply one of the above rules, it reduces the number of untrusted elements by one. It also adds logic to our PPE Circuit, sometimes including 2 to 4 conditional branches. The running time of our algorithm is exponential in the worst case (see Section 3.4), but as discussed above, our tool is surprisingly efficient in practice. In our tool, we also added optimizations to identify and reduce redundant logic, where some branches led to checking the same values repeatedly. In our tests, these optimizations greatly reduced the size of the output circuits. The example we used here with $T = \{g^a, g^b\}$ and $U = \{g^{ab}, g^{ac}, g^{ac/(a+b)}, g^{1/(d+a)}\}$ uses all four rules and after optimizations results in a PPE Circuit with 6 PPEs and 8 boolean gates. It is a simplified version of the Custom Test Case 5 in Table 1 and Appendix D.2.

1.2.1 *Relationship to Prior Work.*

On the same inputs, the running time of our tool and the most comprehensive prior tool called AutoCircuitPPE [47] are usually within a few seconds of each other. They also find similar solutions.[4] This is good news considering

[3] This combination is sometimes as simple as C AND C', but as the reader can see from Figure 7, it may also be more complex.

[4] Currently, both tools output the first solution they find instead of caching several solutions and picking the most optimal. We discuss the possibility of adjusting this for our tool in Section 4.4.

that our support for rational polynomials considerably enlarges the potential branching degree of the search algorithm, which could have negatively impacted both the search time and PPE Circuit size. Our tool is even faster in some cases (such as the 100-DDH test case).

The AutoRationalPPE code uses some elements of the public source code of AutoCircuitPPE[2]. Both tools use the Generic Group Analyzer [19], which has some support for rational polynomials. However, we chose to write the needed subroutines to handle rational polynomials from scratch.

The high-level idea behind our Rules 1 and 2 have roots in similar rules from [20, 46, 47], but are more complex as they require "tracking zeros" as discussed above to prevent elements becoming undefined. The high-level idea behind our Rules 3a and 3b was inspired by [46, 47], although we removed redundancy, added generalization and expanded it to keep elements from becoming undefined. We then split it in two depending on whether a variable not in InTrusted appears in the numerator (Rule 3a) or the denominator (Rule 3b).

2 DEFINITIONS: EXPANDING PPE CIRCUITS

In the section, we formalize the notion of PPE Circuits with support for rational polynomials. It is an extension of the basic PPE Circuits proposed in [47], and the basic PPE instance and challenge notions proposed in [46]. We make a few critical changes needed to handle undefined elements arising from an evaluation of zero in the denominator of a rational polynomial. Let g_1, g_2, g_T be group generators of prime order p for groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively. Following [46, 47], we first rewrite any cryptographic scheme using a single group generator for each group. For example, all elements in $\mathbb{G}_1$ are set up to be derived as g_1^x for a single generator $g_1 \in \mathbb{G}_1$ and $x \in \mathbb{Z}_p$. Thus, we now represent each group element in the scheme with its group, along with a polynomial representing its exponent. We represent a pairing-based cryptographic scheme in this form as a PPE problem instance.

Definition 2.1 (PPE Problem Instance [46, 47]). *A* pairing product equation (PPE) problem instance Π *consists of*[5]

- *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 poly.* $\mathbf{r} = (f_1/h_1, \ldots, f_m/h_m)$ *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* Trusted $\subseteq [1, m]$.

The pairing parameters above can optionally indicate the type of pairing group (e.g., Type I, II or III); unless otherwise specified we assume Type III pairings. Throughout the paper, we use the notation InTrusted(Π) to denote the set of variables that appear in the Trusted set of polynomials of Π i.e., InTrusted(Π) = $\cup_{i \in \text{Trusted}}\{\text{variables used in } f_i\} \cup \{\text{variables used in } h_i\} \subseteq \mathbf{u}$. We simplify the notation and use InTrusted whenever the problem instance Π is implicit.

Definition 2.2 (PPE Challenge [46]). *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{r} = (f_1/h_1, \ldots, f_m/h_m), \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ *be a PPE problem instance as in Definition 2.1. Let* $\mathbf{R} = (R_1, \ldots, R_m)$ *be comprised of pairing group elements, where each* R_i *is in group* $\mathbb{G}_{\alpha_i}$. $\mathbf{R}$ *is called a* challenge *to PPE instance* Π. *Challenges are classified as:*

- $\mathbf{R} = (R_1, \ldots, R_m)$ *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, $R_i = g_{\alpha_i}^{f_i(\mathbf{u})/h_i(\mathbf{u})}$.[6]
- $\mathbf{R} = (R_1, \ldots, R_m)$ *is a* NO *challenge if it is not a* YES *challenge* and $\exists$ *an assignment to* $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}_p^n$ *such that for all* $i \in$ Trusted, $R_i = g_{\alpha_i}^{f_i(\mathbf{u})/h_i(\mathbf{u})}$.[6]
- $\mathbf{R} = (R_1, \ldots, R_m)$ *is an* INVALID *challenge if it is neither a* YES *nor* NO *challenge.*

Following [46, 47], we can view an YES challenge as meaning that both the trusted and untrusted elements are distributed as they should be. Whereas in a NO challenge, the trusted elements are correctly formed, but the untrusted ones are not. In an INVALID challenge, the "trusted" elements are not drawn from the proper distribution (e.g., the public parameters are not correct). Thus, we do not consider this case.

Our goal is to automatically generate circuits that take as input a *PPE challenge* (Definition 2.2) and output 1 for all YES challenges and 0 for all NO challenges. The circuit will input a set of pairing elements and output a single bit. Each gate of the circuit can be an AND/OR/NOT/PPE gate.

The following three definitions do not require alteration from [47].

Definition 2.3 (PPE Circuit [47]). *A* PPE circuit C *is a tuple* $(\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}, \text{out}, \text{GateType}, A, B)$, *where*

- $\mathcal{G} = (p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *establishes the algebraic setting,*
- *integer* m *specifies the number of group elements in the circuit input. We will refer to these as* Inputs $= \{1, \ldots, m\}$.
- *the vector* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ *is a sequence of pairing group identifiers in* $\mathcal{I} = \{1, 2, T\}$ *for the input elements,*
- *integer* N *is the number of gates in the PPE circuit,*
- Gates $= \{m+1, \ldots, m+N\}$. *We will refer to* Wires $=$ Inputs $\cup$ Gates.
- out *is the integer in* Gates *denoting the output gate. Unless otherwise stated,* out $= m + N$.
- GateType : Gates $\rightarrow$ {PPE, AND, OR, NOT} *is a function that identifies the gate functionality, which is one of the following:*
  - PPE*: description includes a circuit with* m Inputs *wires whose logic forms that of a PPE over variables* $R_1, \ldots, R_m$ *where each* $R_i \in \mathbb{G}_{\alpha_i}$ *as specified by* $\boldsymbol{\alpha}$ *and the single output wire of the PPE carries a bit representing whether or not the input satisfies the PPE,*
  - AND*: for gate* g, *the description specifies two integers* a, b *where* $m + 1 \leq a < b < g$.
  - OR*: for gate* g, *the description specifies two integers* a, b *where* $m + 1 \leq a < b < g$.

[5] Unlike the definition of [46], we do not include the set Fixed in the PPE Problem Instance definition and unlike [47], we allow rational polynomials.

[6] Note that since R_i is a well-defined group element, satisfying $R_i = g_{\alpha_i}^{f_i(\mathbf{u})/h_i(\mathbf{u})}$ also implies $h_i(\mathbf{u}) \neq 0$.

– NOT: *for gate g, the description specifies one integer a where $m+1 \le a < g$.*

- *A : Gates $\to$ Wires and B : Gates $\to$ Wires are functions. For any gate* AND/OR/NOT *g, $A(g)$ identifies g's first incoming wire. For any* AND/OR *gate g, $B(g)$ identifies g's second incoming wire. We require that $g > B(g) > A(g)$, ignoring $B(g)$ when undefined. Recall that the input wires for all* PPE *gates are the* Inputs.

The circuit takes as input m group elements and outputs a single output on a wire out.

Definition 2.4 (PPE Circuit Evaluation [47]). *A* PPE circuit evaluation $\mathsf{Eval} : C \times (x_1, \ldots, x_m)$ *takes as input a PPE circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \mathsf{Gates}, \mathsf{out}, \mathsf{GateType}, A, B)$ together with an m-element PPE challenge $(x_1, \ldots, x_m)$ which must be consistent with $(\mathcal{G}, \boldsymbol{\alpha})$ (i.e., $x_i \in \mathbb{G}_{\alpha_i}$). The algorithm outputs a bit in $\{0,1\}$. The default evaluation algorithm is as follows. The input group elements $(x_1, \cdots, x_m)$ are assigned to the m input wires. For each gate $g \in$ Gates (in the increasing order of g), compute s_g as follows according to the description of* $\mathsf{GateType}(g)$*:*

- *if* (PPE, β)*, then evaluate the PPE β using the assignment to variables in $(R_1, \ldots, R_k)$. If the PPE is satisfied, then set $s_g = 1$. Otherwise, set $s_g = 0$.*
- *if* AND*, then $s_g = s_{A(g)} \wedge s_{B(g)}$.*
- *if* OR*, then $s_g = s_{A(g)} \vee s_{B(g)}$.*
- *if* NOT*, then $s_g = \neg s_{A(g)}$.*

This algorithm outputs s_{out}. For the AND, OR *and* NOT *gates, by the rules of the circuit description, $s_{A(g)}$ and $s_{B(g)}$ will be defined before they are used.*

Following [47], we let $C(\mathbf{x})$ denote $\mathsf{Eval}(C, \mathbf{x})$ i.e., evaluation of the circuit C on input $\mathbf{x}$. We let $C_g(\mathbf{x})$ denote the output of gate g of the circuit C on input $\mathbf{x}$.

Definition 2.5 (PPE Circuit Testable / Testing Circuits [47]). *A PPE problem instance $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \mathsf{Trusted})$ is said to be* PPE circuit testable *if and only if there exists a PPE circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \mathsf{Gates}, \mathsf{out}, \mathsf{GateType}, A, B)$. such that both of the following hold:*

- *$C(\mathbf{x}) = 1$ for every* YES *challenge $\mathbf{x}$,*
- *$C(\mathbf{y}) = 0$ for every* NO *challenge $\mathbf{y}$.*

There are no conditions on the behavior of C for INVALID *challenges. For any PPE problem instance Π, we call such a PPE circuit C a* testing circuit. *A testing circuit for a PPE problem instance need not be unique.*

For consistency, we adopt the circuit shorthands from [47], which we review for the reader in Appendix B.

3 SEARCHING FOR A PPE TESTING CIRCUIT WITH RATIONAL POLYNOMIAL SUPPORT

In this section, we describe an algorithm that searches for a PPE testing circuit Q for a given PPE problem. The algorithm takes a PPE problem Π as input and either outputs a PPE testing circuit Q or the special symbol unknown. In the former case, the problem Π is circuit testable. In contrast, in the latter case, we cannot determine whether Π is PPE circuit testable or not. Therefore, the algorithm has one-sided correctness. If the algorithm outputs that Π has testing circuit Q, this will be true.

Following the prior works [46, 47], our algorithm proceeds in a sequence of steps. In each step, the algorithm (attempts to) "reduce the complexity" of its input by adding a rational polynomial f_i/h_i to the set Trusted and simultaneously modifying the testing circuit Q. The prior works define a set of rules to determine which polynomial is supposed to be added to Trusted and how to modify the testing circuit Q at each step. In the end, if we can obtain $\mathsf{Trusted} = [1, m]$, then we will have found a testing circuit. If, at any point, $\mathsf{Trusted} \neq [1, m]$ but none of the movement rules can be applied, the algorithm terminates and outputs unknown. As the earlier works consider only regular polynomials, we extend the rules to the case where the PPE problem contains rational polynomials.

Reception List

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

(1) for each $i \in \{1, 2, T\}$, initialize $\mathbf{l}_i$ with $|\mathbf{t}_i|$ number of fresh variables, i.e., let $\mathbf{l}_i = \{w_{i,1}, \cdots, w_{i,|\mathbf{l}_i|}\}$
(2) If an isomorphism $\psi : \mathbb{G}_1 \to \mathbb{G}_2$ exists, then $\mathbf{l}_2 := \mathbf{l}_2 \cup \mathbf{l}_1$. If an isomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$ exists, then $\mathbf{l}_1 := \mathbf{l}_1 \cup \mathbf{l}_2$
(3) $\mathbf{l}_T := \mathbf{l}_T \cup \{\beta_1\beta_2 : \beta_1 \in \mathbf{l}_1, \beta_2 \in \mathbf{l}_2\}$

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

3.1 Completion Lists for a List of Polynomials

In the section, we review the concept of completion lists in the pairing setting as described by Barthe et al. [19]. Consider any list $\mathbf{r} = [f_1/h_1, \ldots, f_k/h_k]$ of polynomials. Let the i^{th} entry belongs to the group $\mathbb{G}_{\alpha_i}$, where $\alpha_i \in \mathcal{I} = \{1, 2, T\}$ for all $i \le k$. For any group $\mathbb{G}_i$, let $\mathbf{t}_i$ be all the polynomials in the group i.e., $\mathbf{t}_i = \{f_j/h_j : \alpha_j = i\}$. We now recall the notion of completion $\mathsf{CL}(\mathbf{r}) = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\}$ of the list $\mathbf{r}$ of polynomials with respect to a group setting [19]. Intuitively, $\mathsf{CL}(\mathbf{r})$ 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 the list $\mathbf{r}$.

The algorithm to compute the completion $\mathsf{CL}(\mathbf{r})$ proceeds in two steps. In the first step, it computes the reception lists $\{\mathbf{l}_i\}_{i \in \mathcal{I}}$ as shown in Figure 1. The elements of the reception lists are monomials over variables $w_{i,j}$ for $i \in \mathcal{I}$, $j \in |\mathbf{t}_i|$. 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, the algorithm computes the actual polynomials from the reception lists as $\mathbf{s}_i = [m_1(\mathbf{t}), \ldots, m_{|\mathbf{l}_i|}(\mathbf{t})]$ for $[m_1, \ldots, m_{|\mathbf{l}_i|}] = \mathbf{l}_i$, 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 $j \in |\mathbf{t}_i|$. When evaluating these monomials, we do not cancel out any common factor in the numerator and the denominator of the result.

3.2 Rules for Moving Polynomials into the Trusted Set

We now describe four rules for reducing the complexity of a PPE instance. We mean reducing the number of elements represented

by the rational polynomials, not in the set Trusted. We derive the rules closely based on the rules proposed in [46, 47] that were designed for non-rational polynomials. From now on, we assume the formal variables $R_1, R_2, \cdots, R_m$ represent group elements of any PPE challenge corresponding to Π. These formal variables also represent the input wires of the PPE circuit C being constructed.

3.2.1 **Rule 1: Simple move of a rational polynomial with all** InTrusted **variables to** Trusted **set.** In this section, we show how to simplify the given PPE problem by moving a rational polynomial not in Trusted to Trusted. Given a PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and an index $k \in [m]$, rule 1 can possibly be applied if $k \notin$ Trusted and the polynomial $f_k/h_k \in \mathbf{r}$ consists only of variables $u_i \in$ InTrusted (these conditions are necessary, but not sufficient). The rule 1, which is shown in Figure 2 is adapted from Rule 1 in [46, 47]. These works for non-rational polynomials express the untrusted polynomial f_k in terms of polynomials in Trusted. Such an expression gives rise to a pairing product equation that can verify the well-formedness of the k^{th} element in any PPE challenge. In this paper, we adopt Rule 1 to rational polynomials. Here, we additionally express the denominator h_k in terms of the Trusted polynomials and add a pairing production equation to the final PPE circuit to verify that the denominator h_k does not evaluate to 0. We now formally describe our Rule 1 in Figure 2 and prove its correctness property in Lemma 3.1.

Description of Rule 1

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

(1) If $k \in$ Trusted or $r_k = f_k/h_k \in \mathbf{r}$ has variables not in InTrusted, abort and output $\perp$.
(2) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{r}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \le |\mathbf{s}_i|$, let $\mathbf{s}_i[j] = \overline{f}_i[j]/\overline{h}_i[j]$, $\mathbf{S}_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$, and let $U_i[j]$ be the pairing product term computing $\mathbf{S}_i[j]$ in terms of formal variables $R_1, \cdots, R_m$.
(3) For each $i \in \mathcal{I}$, let H_i be a least common multiple of the polynomials $\{\overline{h}_i[j]\}_{j \in |\mathbf{s}_i|}$, and let the polynomial $\hat{f}_i[j]$ be such that $\hat{f}_i[j]/H_i \equiv \overline{f}_i[j]/\overline{h}_i[j]$.
(4) Check if there exist and index $i \in \mathcal{I}$ and constant vectors $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{s}_T|})$ and $\mathbf{b} = (b_1, \cdots, b_{|\mathbf{s}_i|})$ with entries in $\mathbb{Z}_p$ s.t.

$$r_k \equiv \frac{f_k}{h_k} \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j] \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \frac{\hat{f}_T[j]}{H_T} \tag{1}$$

$$h_k \equiv \sum_{j=1}^{|\mathbf{s}_i|} b_j \cdot \mathbf{s}_i[j] \equiv \sum_{j=1}^{|\mathbf{s}_i|} b_j \cdot \frac{\hat{f}_i[j]}{H_i} \tag{2}$$

These equations can also be expressed as

$$f_k \cdot H_T \equiv h_k \cdot \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \hat{f}_T[j] \tag{3}$$

$$h_k \cdot H_i \equiv \sum_{j=1}^{|\mathbf{s}_i|} b_j \cdot \hat{f}_i[j] \tag{4}$$

respectively. Computing such coefficient vectors reduces to checking if the polynomial $\mathbf{0}$ belongs to the span of all the polynomials in the left hand side and the right hand side of the above equation.
(5) If such $\mathbf{a}$, $\mathbf{b}$ exists, define the PPEs

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

$$B := \prod_{j=1}^{|\mathbf{s}_i|} U_i[j]^{b_j} = \mathbb{I}_i$$

where $\mathbb{I}_i$ is the identity element in group $\mathbb{G}_i$. Now define PPE circuit C as MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, A) AND (NOT MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, B)). Output the circuit C along with PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$. If such $\mathbf{a}$, $\mathbf{b}$ do not exist, output $\perp$.

Figure 2: Procedure for moving certain rational polynomials $r_k = f_k/h_k$ with all InTrusted **variables to** Trusted

Lemma 3.1 (Correctness of Rule 1). *Let $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ be a PPE problem instance as in Definition 2.1 and let $k \in [m]$. Suppose $\perp \neq (C, \Pi') = \text{Rule1}(\Pi, k)$. Then, for every testing circuit C' for Π', it holds that C AND C' is a testing circuit for Π.*

Proof. The proof of this lemma appears in Appendix C.1.

Description of SubstituteZero **Algorithm**

Input: A PPE Problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) and polynomial h.

Output: A PPE Problem Π' or $\perp$.

- Construct vector $\mathbf{r}'$ of m rational polynomials as follows. For each $k \in [m]$, let $r_k = f_k/h_k$.
  - If $f_k = \mathsf{Poly}_1 \cdot h + \mathsf{Poly}_2$ for some polynomials Poly_1 and Poly_2, such that (1) Poly_2 does not have h as a factor, and (2) the number of monomials in Poly_2 is less than f_k when expressed in canonical form, then set $\mathsf{num}_k = \mathsf{Poly}_2$. Otherwise, set $\mathsf{num}_k = f_k$.
  - If $h_k = \mathsf{Poly}_3 \cdot h + \mathsf{Poly}_4$ for some polynomials Poly_3 and Poly_4, such that (1) Poly_3 does not have h as a factor, and (2) the number of monomials in Poly_4 is less than h_k when expressed in canonical form, then set $\mathsf{denom}_k = \mathsf{Poly}_4$. Otherwise, set $\mathsf{denom}_k = h_k$.
  - $r'_k = \mathsf{num}_k/\mathsf{denom}_k$.
- If $\mathsf{denom}_k = 0$ for any k, then output $\perp$. Otherwise, output $\Pi' = (\mathcal{G}, n, m, \mathbf{r}', \mathbf{u}, \boldsymbol{\alpha}$, Trusted).

Figure 3: Algorithm for updating a PPE problem instance when a specified polynomial h is set to 0.

3.2.2 **Rule 2: More general move of a rational polynomial with all** InTrusted **variables to** Trusted **set.** In this section, we show a more general way to move a rational polynomial not in Trusted to Trusted. Given a PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) and an index $k \in [m]$, rule 2 can possibly be applied if $k \notin$ Trusted and the polynomial $f_k/h_k \in \mathbf{r}$ consists only of variables $u_i \in$ InTrusted (these conditions are necessary, but not sufficient). In Rule 1, we expressed the untrusted polynomial f_k/h_k in terms of polynomials in Trusted. However, in the expression, we didn't allow f_k/h_k to be multiplied by any factor. In rule 2, we consider more general way to express f_k/h_k in terms of polynomials in Trusted, by allowing expressions of the form

$$(f_k/h_k) \cdot (\text{some combination of trusted polynomials})$$

$$= (\text{some other combination of trusted polynomials}).$$

Once we obtain such an expression, we move f_k/h_k to the trusted set and add a PPE corresponding to the expression to our final PPE circuit. This PPE is supposed to verify well-formedness of k^{th} untrusted element in any PPE challenge.

However, there is one issue here. Suppose the factor that is multiplied to f_k/h_k in the above expression evaluates to 0 on a given PPE challenge. In that case, the PPE does not verify the well-formedness of f_k/h_k as the PPE might be trivially satisfied. To solve the issue, we adopt the approach proposed by [46, 47] in their Rule 3. We break the scenario into 2 cases. (1) The factor that is multiplied to f_k/h_k in the above expression does not evaluate to 0. (2) The factor evaluates to 0. In the former case, the above PPE validates the well-formedness of the untrusted element. In the latter case, we try to apply other rules. As earlier, we additionally express the denominator h_k in terms of the Trusted polynomials and add a pairing production equation to the final PPE circuit to verify that the denominator h_k does not evaluate to 0. We now formally describe our Rule 2 in Figure 4 and prove its correctness property in Lemma 3.2.

Lemma 3.2 (Correctness of Rule 2). *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) *be a PPE problem instance as in Definition 2.1 and let* $k \in [m]$. *Suppose* $\perp \neq (\mathsf{IsIdentity}, C, \Pi', \Pi'') = \mathsf{Rule2}(\Pi, k)$.

- *If* $\Pi'' \neq \perp$, *for every pair of testing circuits* C' *and* C'' *for* Π' *and* Π'' *respectively, the PPE circuit*

  $Z :=$ ((NOT IsIdentity) AND C AND C') OR (IsIdentity AND C'')

  is a testing circuit for Π.
- *If* $\Pi'' = \perp$, *for every testing circuit* C' *for* Π',

  $$Z := ((\text{NOT IsIdentity}) \text{ AND } C \text{ AND } C')$$

  is a testing circuit for Π.

Proof. The proof of this lemma appears in Appendix C.2.

3.2.3 **Rule 3a: General move of a rational polynomial $r_k = f_k/h_k$ with multiple non-**InTrusted **variables to the** Trusted **set.** We now describe a way to move a rational polynomial not in Trusted to Trusted when the polynomial is allowed to have non-InTrusted variables[7] in the numerator. Given a PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) and an index $k \in [m]$, rule 3a can possibly be applied if $k \notin$ Trusted, the polynomial $h_k \in \mathbf{r}$ consists only of variables $u_i \in$ InTrusted, and f_k contains one or more non-InTrusted variables (these conditions are necessary, but not sufficient). In Figure 5, we formally describe the Rule 3a, which is an extension of Rule 4 in [46, 47]. We prove its correctness property in Lemma 3.3.

Lemma 3.3 (Correctness of Rule 3a). *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) *be a PPE problem instance as in Definition 2.1,* $j \in [n]$ *and* $k \in [m]$. *Suppose* $\perp \neq (\mathsf{IsIdentity}, C, \Pi', \Pi'') = \mathsf{Rule3a}(\Pi, j, k)$.

- *If* $\Pi'' \neq \perp$, *for every pair of testing circuits* C' *and* C'' *for* Π' *and* Π'' *respectively, the PPE circuit*

  $Z :=$ ((NOT IsIdentity) AND C AND C') OR (IsIdentity AND C'')

  is a testing circuit for Π.
- *If* $\Pi'' = \perp$, *for every testing circuit* C' *for* Π',

  $$Z := ((\text{NOT IsIdentity}) \text{ AND } C \text{ AND } C')$$

  is a testing circuit for Π.

Proof. The proof of this lemma appears in Appendix C.3.

3.2.4 **Rule 3b: General move of a rational polynomial $r_k = f_k/h_k$ with multiple non-**InTrusted **variables to the** Trusted **set.** In this section, we show describe a way to move a rational polynomial not in Trusted to Trusted when the polynomial is allowed to have non-InTrusted variables[8] in the denominator. Given a PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) and an index $k \in [m]$, rule 3b can possibly be applied if $k \notin$ Trusted, the polynomial $f_k \in \mathbf{r}$ consists only of variables $u_i \in$ InTrusted, and h_k contains one or more non-InTrusted variables (these conditions are necessary, but not sufficient). In Figure 6, we formally describe the Rule 3b. We prove its correctness property in Lemma 3.4.

[7] Recall that InTrusted variables are the set of all variables used in the Trusted set of polynomials.

[8] Recall that InTrusted variables are the set of all variables used in the Trusted set of polynomials.

Description of Rule 2

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and an integer $k \in [1, m]$.

Output: Two PPE circuits IsIdentity, C and two circuit PPE problems Π', Π'', or the symbol $\perp$ (meaning could not apply rule).

Steps of Rule2(Π, k):

(1) If $k \in$ Trusted or $\alpha_k = T$ or $r_k = f_k/h_k \in \mathbf{r}$ has variables not in InTrusted, abort and output $\perp$.

(2) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \text{CL}(\mathbf{r}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \le |\mathbf{s}_i|$, let $\mathbf{s}_i[j] = \overline{f}_i[j]/\overline{h}_i[j]$, $\mathbf{S}_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$, and let $U_i[j]$ be the pairing product term computing $\mathbf{S}_i[j]$ in terms of formal variables $R_1, \cdots, R_m$.

(3) For each $i \in \mathcal{I}$, let H_i be a least common multiple of the polynomials $\{\overline{h}_i[j]\}_{j \in |\mathbf{s}_i|}$, and let the polynomial $\hat{f}_i[j]$ be such that $\hat{f}_i[j]/H_i \equiv \overline{f}_i[j]/\overline{h}_i[j]$.

(4) Let $\alpha = 3 - \alpha_k$. Check if there exists an index $i \in \mathcal{I}$ and constant vectors $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{s}_T|})$, $\mathbf{b} = (b_1, \cdots, b_{|\mathbf{s}_\alpha|})$ and $\mathbf{c} = (c_1, \cdots, c_{|\mathbf{s}_i|})$ with entries in $\mathbb{Z}_p$ s.t.

$$r_k \cdot (\sum_{j=1}^{|\mathbf{s}_\alpha|} b_j \cdot \mathbf{s}_\alpha[j]) \equiv \frac{f_k}{h_k} \cdot \Big(\sum_{j=1}^{|\mathbf{s}_\alpha|} b_j \cdot \frac{\hat{f}_\alpha[j]}{H_\alpha}\Big) \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j] \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \frac{\hat{f}_T[j]}{H_T} \tag{5}$$

$$h_k \equiv \sum_{j=1}^{|\mathbf{s}_i|} c_j \cdot \mathbf{s}_i[j] \equiv \sum_{j=1}^{|\mathbf{s}_i|} c_j \cdot \frac{\hat{f}_i[j]}{H_i} \tag{6}$$

These equations can also be expressed as

$$f_k \cdot H_T \cdot (\sum_{j=1}^{|\mathbf{s}_\alpha|} b_j \cdot \hat{f}_\alpha[j]) \equiv h_k \cdot H_\alpha \cdot \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \hat{f}_T[j] \tag{7}$$

$$h_k \cdot H_T \equiv \sum_{j=1}^{|\mathbf{s}_T|} c_j \cdot \hat{f}_T[j] \tag{8}$$

(Computing coefficient vectors **a**, **b**, **c** reduces to checking if the polynomial **0** belongs to the span of all the polynomials in the left-hand side and the right-hand side of the above equation.)

(5) If such (**a**, **b**, **c**) exist, then compute PPEs

$$A := (\prod_{j=1}^{|\mathbf{s}_\alpha|} U_\alpha[j]^{b_j} = \mathbf{I}_\alpha), \qquad D := \mathbf{I}_i = \prod_{j=1}^{|\mathbf{s}_i|} U_i[j]^{c_j} \qquad B := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} = \begin{cases} e(R_k, \prod_{j=1}^{|\mathbf{s}_\alpha|} U_\alpha[j]^{b_j}) & \text{if } \alpha_k = 1 \\ e(\prod_{j=1}^{|\mathbf{s}_\alpha|} U_\alpha[j]^{b_j}, R_k) & \text{if } \alpha_k = 2 \end{cases}$$

where $\mathbf{I}_\alpha$ is the identity element in group $\mathbb{G}_\alpha$.

- Compute $\Pi' = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$ and $\Pi'' = \text{SubstituteZero}(\Pi, \sum_{j=1}^{|\mathbf{s}_\alpha|} b_j \cdot \hat{f}_\alpha[j])$, where the SubstituteZero algorithm is described in Figure 3. Intuitively, SubstituteZero creates a new PPE problem instance by substituting $\sum_{j=1}^{|\mathbf{s}_\alpha|} b_j \cdot \hat{f}_\alpha[j]$ with 0 in the Trusted set of polynomials).
- If $\Pi'' = \Pi$, then output $\perp$. Otherwise, output the circuit IsIdentity := MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, A), the circuit C := MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, B) AND (NOT MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, D)) and PPE problems Π', Π''.

(6) If such (**a**, **b**, **c**) do not exist, then output $\perp$.

Figure 4: A more general procedure for moving certain rational polynomials $r_k = f_k/h_k$ with all InTrusted variables to Trusted

Lemma 3.4 (Correctness of Rule 3b). *Let $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ be a PPE problem instance as in Definition 2.1, $j \in [n]$ and $k \in [m]$. Suppose $\perp \neq (\text{IsIdentity}, C, \Pi', \Pi'') = \text{Rule3b}(\Pi, j, k)$.*

- *If $\Pi'' \neq \perp$, for every pair of testing circuits C' and C'' for Π' and Π'' respectively, the PPE circuit*

  $Z :=$ ((NOT IsIdentity) AND C AND C') OR (IsIdentity AND C'')

  is a testing circuit for Π.
- *If $\Pi'' = \perp$, for every testing circuit C' for Π',*

  $$Z := ((\text{NOT IsIdentity}) \text{ AND } C \text{ AND } C')$$

  is a testing circuit for Π.

Proof. The proof of this lemma appears in Appendix C.4.

3.3 Applying the Rules

We now describe how to combine Rules 1-3b into the main algorithm that takes input a PPE problem and outputs a PPE circuit or the special message unknown. Here unknown means that the search did not produce an output but does not prove that no such testing circuit for the input problem exists. We describe the algorithm QSearch in Figure 7. Later in Theorem 1, we prove that if this algorithm produces a testing circuit as output, then that circuit is guaranteed to classify PPE challenges for this PPE problem correctly.

Theorem 1 (Correctness of the PPE Circuit Searching Algorithm in Figure 7). *Let $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ be a PPE problem instance as in Definition 2.1. Let $C = \text{QSearch}(\Pi)$. If $C \neq$ unknown, then C is a PPE testing circuit for Π as in Definition 2.5, and therefore Π is circuit testable.*

Proof. This follows the corresponding theorem in [47]. We sketch how to prove this by induction on the number of untrusted polynomials and the total number of monomials in all the polynomials of **f**. The critical correctness arguments required have already been covered for each rule in Lemmas 3.1, 3.2, 3.3, 3.4. When QSearch is invoked on Π with either zero untrusted polynomials or

Description of Rule 3a

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and integer $k \in [m]$.

Output: Either two PPE circuits IsIdentity and C and two PPE problems Π', Π'', or $\perp$ (meaning could not apply the rule).

Steps of Rule3a(Π, j, k):

(1) Let the polynomial $r_k = f_k/h_k \in \mathbf{r}$. If $k \in$ Trusted, then abort and output $\perp$.

(2) If f_k is of the form $h' \cdot u_j^d + h''$ and any of the following conditions do not hold, abort and output $\perp$.
- $j \notin$ InTrusted,
- the polynomials h', h_k contains only variables in InTrusted, and
- the polynomial h'' contains any variables other than u_j, and
- the constant $d \in \mathbb{Z}_p$ is relatively prime to $p-1$,

(3) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{r}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \le |\mathbf{s}_i|$, let $\mathbf{s}_i[j] = \overline{f}_i[j]/\overline{h}_i[j]$, $\mathbf{S}_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$, and let $U_i[j]$ be the pairing product term computing $\mathbf{S}_i[j]$ in terms of formal variables $R_1, \cdots, R_m$.

(4) For each $i \in \mathcal{I}$, let H_i be a least common multiple of the polynomials $\{\overline{h}_i[j]\}_{j \in |\mathbf{s}_i|}$, and let the polynomial $\hat{f}_i[j]$ be such that $\hat{f}_i[j]/H_i \equiv \overline{f}_i[j]/\overline{h}_i[j]$.

(5) Check if there exists constant vectors $\mathbf{a} = (a_1, a_2, \cdots, a_{|\mathbf{s}_T|})$ and $\mathbf{b} = (b_1, b_2, \cdots, b_{|\mathbf{s}_T|})$ with elements in $\mathbb{Z}_p$ such that $h' \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \frac{\hat{f}_T[j]}{H_T}$, and $h_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} b_j \cdot \frac{\hat{f}_T[j]}{H_T}$ These equations can be equivalently expressed as $h' \cdot H_T \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \hat{f}_T[j]$ and $h_k \cdot H_T \equiv \sum_{j=1}^{|\mathbf{s}_T|} b_j \cdot \hat{f}_T[j]$.

(6) If such vector $\mathbf{a}$ exists, then
- Set PPE
$$A := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} = \mathbf{I}_T, \qquad B := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{b_j} = \mathbf{I}_T,$$
where $\mathbf{I}_T$ is the identity element of the group $\mathbb{G}_T$. Define IsIdentity := MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, A), C := NOT MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, B).
- Set PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$ and Π'' = SubstituteZero(Π, h'), where the function SubstituteZero is defined in Figure 3. If $\Pi'' = \Pi$, then output $\perp$. Otherwise, output IsIdentity and Π', Π''.

(7) If such vector $\mathbf{a}$ does not exist, then output $\perp$.

Figure 5: Procedure for moving a rational polynomial $r_k = f_k/h_k$ containing non-InTrusted variables only in f_k to Trusted

Description of Rule 3b

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and integer $k \in [m]$.

Output: Either a PPE circuit IsIdentity and two PPE problems Π', Π'' or $\perp$ (meaning could not apply the rule).

Steps of Rule3b(Π, j, k):

(1) Let the polynomial $r_k = f_k/h_k \in \mathbf{r}$. If $k \in$ Trusted, then abort and output $\perp$.

(2) If h_k is of the form $h' \cdot u_j^d + h''$ and any of the following conditions do not hold, abort and output $\perp$.
- $j \notin$ InTrusted,
- the polynomials h', f_k contains only variables in InTrusted, and
- the polynomial h'' contains any variables other than u_j, and
- the constant $d \in \mathbb{Z}_p$ is relatively prime to $p-1$,

(3) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{r}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \le |\mathbf{s}_i|$, let $\mathbf{s}_i[j] = \overline{f}_i[j]/\overline{h}_i[j]$, $\mathbf{S}_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$, and let $U_i[j]$ be the pairing product term computing $\mathbf{S}_i[j]$ in terms of formal variables $R_1, \cdots, R_m$.

(4) For each $i \in \mathcal{I}$, let H_i be a least common multiple of the polynomials $\{\overline{h}_i[j]\}_{j \in |\mathbf{s}_i|}$, and let the polynomial $\hat{f}_i[j]$ be such that $\hat{f}_i[j]/H_i \equiv \overline{f}_i[j]/\overline{h}_i[j]$.

(5) Check if there exists constant vectors $\mathbf{a} = (a_1, a_2, \cdots, a_{|\mathbf{s}_T|})$ and $\mathbf{b} = (b_1, b_2, \cdots, b_{|\mathbf{s}_T|})$ with elements in $\mathbb{Z}_p$ such that
$$h' \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \frac{\hat{f}_T[j]}{H_T}, \qquad f_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} b_j \cdot \frac{\hat{f}_T[j]}{H_T}.$$
These equations can be equivalently expressed as $h' \cdot H_T \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \hat{f}_T[j]$, and $f_k \cdot H_T \equiv \sum_{j=1}^{|\mathbf{s}_T|} b_j \cdot \hat{f}_T[j]$.

(6) If such vector $\mathbf{a}$ and $\mathbf{b}$ exists, then
- Set PPE
$$A := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} = \mathbf{I}_T, \qquad B := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{b_j} = \mathbf{I}_T, \qquad D := R_k = \mathbf{I}_T$$
where $\mathbf{I}_T$ is the identity element of the group $\mathbb{G}_T$. Define IsIdentity = MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, A), B' = MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, B) and D' = MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, D).
- Set PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$ and Π'' = SubstituteZero(Π, h'), where the function SubstituteZero is defined in Figure 3. If $\Pi'' = \Pi$, then output $\perp$. Otherwise, output IsIdentity, C = (NOT B') OR (B' AND D') and Π', Π''.

(7) If such vector $\mathbf{a}$ does not exist, then output $\perp$.

Figure 6: Procedure for moving a rational polynomial $r_k = f_k/h_k$ containing non-InTrusted variables only in h_k to Trusted

zero total number of monomials, it outputs the always accepting circuit C_{acc} which is a valid testing circuit. Now suppose the QSearch algorithm outputs a valid testing circuit or unknown on every problem Π' which has at most α number of untrusted polynomials and at most β total number of monomials in $\mathbf{f}$. Suppose QSearch outputs a circuit $C \neq$ unknown on a problem Π with $\alpha + 1$ untrusted polynomials and at most β total number of monomials in $\mathbf{r}$. It must have invoked one of the 4 rules. By Lemmas 3.1, 3.2, 3.3, 3.4 and our induction hypothesis, C is a valid testing circuit. Similarly, QSearch outputs either a valid testing circuit or unknown when invoked on a problem Π with at most α untrusted polynomials and $\beta + 1$ total number of monomials in $\mathbf{f}$. By induction, for any Π, if QSearch(Π) does not output unknown, then it outputs a valid testing circuit for Π. ∎

3.4 Efficiency of QSearch

The asymptotic time complexity of the QSearch algorithm will be exponential, although fortunately our experiments from Section 4 show that it is surprisingly fast in practice. Let us now analyze its running time. A call to QSearch scans all the untrusted polynomials to check if any rule is applicable and then calls QSearch recursively at most two times.

Let us first compute the time taken to scan all the untrusted polynomials and check if any rule is applicable. 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 maximum size of all polynomials $\mathbf{f}$ in the input be s. Executing any rule involves computing completion lists followed by checking if $\mathbf{0}$ lies in the span of certain polynomials. Computing completion lists of m polynomials involves $O(m^2)$ polynomial multiplications take $O(m^2 \cdot s^2)$ time. Normalizing the rational polynomials to have a common denominator involves multiplying m^2 polynomials in the completion list each of size s^2, which takes $O(m^2 \cdot s^{2m^2})$ time. Suppose we want to check if $\mathbf{0}$ lies in the span of $O(m^2)$ polynomials (number of polynomials in the completion lists), each having at most $O(s^{2m^2})$ monomials after normalization. This involves solving a system of $O(m^2 \cdot s^{2m^2})$ linear equations (upper bound on the number of monomials in the completion list) each of size $O(m^2)$. This takes at most $O((m^2 \cdot s^{2m^2})^\omega)$ time, where n^ω is the complexity of multiplying two $n \times n$ matrices. Therefore, applying all the rules to all the untrusted polynomials takes at most $O(m \cdot (m^2 \cdot s^{2m^2})^\omega)$ time.

Now let us compute the total number of times we call the QSearch algorithm recursively. Suppose QSearch is run on problem Π and suppose it triggers a rule that outputs two PPE problems Π' and Π''. We obtain the problem Π' by moving an untrusted polynomial to the trusted set, and the problem Π'' is obtained by substituting some polynomial by zero. Note that Π'' cannot be equal to the original problem as Rule2 − 3b outputs $\perp$ otherwise. Therefore some polynomial of Π'' has at least one lesser monomial than Π. Let the total number of monomials in all the polynomials of Π be k. By the above analysis, the QSearch is recursively invoked at most 2^{m+k} times. As each recursive call takes at most $O(m \cdot (ms)^{2\omega})$ time, the total time taken by our algorithm is $O(m \cdot (m^2 \cdot s^{2m^2})^\omega \cdot 2^{m+k})$ time.

Even though our algorithm has high theoretical complexity, in Section 4 we show that it runs reasonably fast for many real-world schemes.

Main Algorithm for PPE Testing Circuit Search

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{r}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$.
Output: A PPE circuit Q or the special symbol unknown.
Steps of QSearch(Π):

Start. If Trusted = $[m]$, then output the always accepting circuit $Q := C_{\text{acc}}$.
Rule 1. For $k = 1$ to m,
(a) Call $z =$ Rule1(Π, k).
(b) If $z = (C, \Pi') \neq \perp$, then
(i) Call $C' =$ QSearch(Π')
(ii) If $C' \neq$ unknown, then output the PPE circuit $Q :=$ C AND C'.
Rule 2-3b. For rule in {Rule2, Rule3a, Rule3b}, $k = 1$ to m,
(a) Call $z =$ rule(Π, k).
(b) If $z =$ (IsIdentity, C, Π', Π'') $\neq \perp$, then
(i) Call $C' =$ QSearch(Π')
(ii) If $C' \neq$ unknown and $\Pi'' = \perp$, then output $Q :=$ ((NOT IsIdentity) AND C AND C').
(iii) If $C' \neq$ unknown and $\Pi'' \neq \perp$, then call $C'' =$ QSearch(Π'')
(iv) If $C' \neq$ unknown and $C'' \neq$ unknown, then output the PPE circuit $Q :=$ ((NOT IsIdentity) AND C AND C') OR (IsIdentity AND C'').
Final. Otherwise, output unknown.

Figure 7: Recursive procedure for searching for a PPE Testing Circuit

4 IMPLEMENTATION

We implemented the PPE circuit searching algorithm described in Figure 7 in a software tool called AutoRationalPPE. We ran the tool on a number of signature, verifiable random function and advanced encryption schemes as well as other types of pairing-based public/private parameters, including some that are PPE circuit testable and some that are provably not. Our tool was able to produce outputs for the schemes based on rational polynomials left open by the previous AutoPPE and AutoCircuitPPE tools [46, 47] and for several new schemes. We now present the design of the AutoRationalPPE tool followed by its test case results and performance numbers.

4.1 AutoRationalPPE Implementation

We implemented AutoRationalPPE using Ocaml version 4.02.3. We built the code on top of the AutoCircuitPPE[9] tool (Hohenberger et al. [47]), which in turn utilizes some of the parsing tools and data structures (to store polynomials) of the Generic Group Analyzer (GGA) tool[10] of Barthe et al. [19]. We also used the SageMath package[11] to solve systems of linear equations and implemented the remaining logic ourselves.

[9] https://github.com/JHUISI/auto-tools
[10] https://github.com/generic-group-analyzer/gga
[11] https://www.sagemath.org/

Input File Example

```
maps G1 * G1 ->GT.
trusted_polys [F1 = a] in G1.
untrusted_polys [F2 = a*a, F3 = a*a*a, F4 = 1/(a*a*a*a), F5 = 1/a] in G1.
```

Figure 8: Input file for our detailed example.

The input format of AutoRationalPPE is similar to the AutoCircuitPPE tool, which makes testing with both tools easier.[12] For the sake of completeness, we present the input format below. The tool's input consists of pairing information (such as the Type I, II or III) and a set of trusted/untrusted polynomials along with their group identifiers.[13] 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 rational polynomials) are bilinear group elements in $\mathbb{G}_1, \mathbb{G}_2$ or $\mathbb{G}_T$ and Definition 2.1 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 similar to AutoPPE and AutoCircuitPPE.[14] The tool runs the algorithm in Figure 7 along with a few optimizations implemented in AutoPPE and AutoCircuitPPE such as computing completion list before applying all the rules. It finally outputs either a PPE circuit or the special symbol unknown. The PPE circuit computed by the QSearch algorithm is generally very large, and therefore we further optimize the circuit by a few techniques such as computing common sub-circuits only once.

The source code for AutoRationalPPE comprises roughly 4K 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 AutoRationalPPE makes the tool highly practical and useful. The source code of AutoRationalPPE is publicly available at https://github.com/JHUISI/auto-tools.

4.2 A Detailed Example

In this section, we explain how to use our tool via a detalied example. In Figure 8, we present a sample input to our tool. Here, we intend to verify the well-formedness of group elements $(g_1^{a^2}, g_1^{a^3}, g_1^{1/a^4}, g_1^{1/a})$, given g_1^a i.e., the a is the trusted polynomial and $(a^2, a^3, 1/a^4, 1/a)$ are the untrusted set of polynomials. We compute the PPE circuit that verifies the well-formedness of above untrusted polynomials using our tool.

In the input file to our tool, we specify the pairing information using the line `maps G1*G1->GT`, which denotes a Type I pairing[15]. We then specify the trusted set of polynomials along with their group identifiers using `trusted_polys [_] in G_`. We then specify the untrusted set of polynomials along with their group identifiers using `untrusted_polys[_] in G_`. For each polynomial, we also specify a formal variable `F_` which is used in the PPE circuit output by the tool. We specify comments using delimeters `(*....*)`. We present the output of the tool on the above input file in Figure 9. The output file contains the list of rules applied during the execution and the final circuit. Following AutoCircuitPPE, we also make a few optimizations to the final circuit. The list of PPE and boolean gates of the optimized circuit are presented at the end of the output. Here, each wire of the circuit is denoted using `Gxx` notation. This could be either output of a PPE or boolean gate.

Output of the Tool

```
F0 = 1 in G1      F0 = 1 in GT      F1 = a in G1      F2 = a^2 in G1
F3 = a^3 in G1    F4 = 1/a^4 in G1  F5 = 1/a in G1

Trusted set in G1: F1 = a
Untrusted set in G1: F2 = a^2, F3 = a^3, F4 = 1/a^4, F5 = 1/a
Rule 1 applied to F2 = a^2.   C := e(F2,F0) = e(F1,F1)

Trusted set in G1: F1 = a, F2 = a^2
Untrusted set in G1: F3 = a^3, F4 = 1/a^4, F5 = 1/a
Rule 1 applied to F3 = a^3.   C := e(F3,F0) = e(F1,F2)

Trusted set in G1: F1 = a, F2 = a^2, F3 = a^3
Untrusted set in G1: F4 = 1/a^4, F5 = 1/a
Rule 2 applied on F5 = 1/a. isidentity := F1 = I   C := (e(F5,F1) = F0 AND
(NOT F1 = I))

Trusted set in G1: F1 = a, F2 = a^2, F3 = a^3, F5 = 1/a
Untrusted set in G1: F4 = 1/a^4
Rule 2 applied on F4 = 1/a^4. isidentity := F2 = I   C := (e(F4,F2) = e(F5,F5)
AND (NOT e(F2,F2) = I))

Execution time : 8.283724s
(e(F2,F0) = e(F1,F1) AND (e(F3,F0) = e(F1,F2) AND (((NOT F1 = I) AND
(e(F5,F1) = F0 AND (NOT F1 = I))) AND ((NOT F2 = I) AND (e(F4,F2) =
e(F5,F5) AND (NOT e(F2,F2) = I))))))

Optimized Circuit:
G1 : e(F2,F0) = e(F1,F1)    G2 : e(F3,F0) = e(F1,F2)    G3 : F1 = I
G4 : e(F5,F1) = F0    G5 : F2 = I    G6 : e(F4,F2) = e(F5,F5)
G7 : e(F2,F2) = I   G8 : NOT G3   G9 : G8 AND G4   G10 : NOT G5
G11 : NOT G7    G12 : G6 AND G11    G13 : G10 AND G12
G14 : G9 AND G13    G15 : G2 AND G14    G16 : G1 AND G15
```

Figure 9: Output file for our detalied example. The final PPE circuit is presented at the end. The wires of the PPE circuit are denoted using Gxx.

In identity based encryption schemes, typically identity id is a variable in $\mathbb{Z}_p$ and not a group element. In such cases, we can specify id variable using `Zp_vars [_]`. Internally for every problem instance Π, for each trusted polynomial f/h and a $\mathbb{Z}_p$ variable x_i, the AutoRationalPPE tool adds $x_i \cdot f/h$ to the trusted set[16]. Prior tools also do this modification.

4.3 Case Studies

We evaluated AutoRationalPPE 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 Figure 1. Like

[12] Unlike AutoCircuitPPE, our tool takes 2 polynomials for each formal variable representing numerator and denominator.

[13] While this program input is in a slightly different format than Definition 2.1, we stress that it is the same information.

[14] Whenever a polynomial f_i/h_i is added to the Trusted set, then the implementation also adds $u_j \cdot f_i/h_i$ for any variables u_j representing elements in $\mathbb{Z}_p$.

[15] 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`.

[16] Ideally, for each polynomial poly on $\mathbb{Z}_p$ variables $\mathbf{x}$, one should include $\text{poly}(\mathbf{x}) \cdot f/h$ in the trusted set. The AutoRationalPPE tool supports such an operation for all bounded degree polynomials on $\mathbb{Z}_p$ variables. However for the purpose of this example, it suffices to include only $x_i \cdot f/h$ to trusted set.

Scheme	Pairing	Type	AutoCircuitPPE output	PPE Circuit Testability	Our Tool Output	#PPE Gates	#Bool Gates	Run Time
BF [29]	Type I	IBE	Testable	Testable	Testable	1	0	3.61s
GS [39]	Type I	IBE	Testable	Testable	Testable	1	0	4.06s
BB [25] ($\ell = 160$)	Type I	HIBE	Testable	Testable	Testable	1	0	177.57s
BB [26] ($\lvert H(\mathsf{id})\rvert = 8$)	Type I	IBE	Testable	Testable	Testable	1	0	25.5s
Waters [56] ($\lvert H(\mathsf{id})\rvert = 16$)	Type I	IBE	Testable	Testable	Testable	1	0	50.22s
N [53] ($\mathcal{B}(H(\mathsf{id})) = 8$)	Type III	IBE	Testable	Testable	Testable	1	0	4.62s
BBG [28] ($\ell = 8$)	Type I	HIBE	Testable	Testable	Testable	5	4	6.87s
Waters [57]	Type I	IBE	Unknown	Not testable	Unknown	0	0	322.23s
BW [31]	Type I	Anon-IBE	Testable	Testable	Testable	28	125	20.56s
BB [25]	Type I	IBE	N/A	Testable	Testable	2	2	5.02s
Gentry [38]	Type I	IBE	N/A	Testable	Testable	2	2	2.15s
BLS [30]	Type I	Signature	Testable	Testable	Testable	1	0	3.69s
CL [33]-A	Type I	Signature	Testable	Testable	Testable	2	1	3.60s
CL [33]-B	Type I	Signature	Testable	Testable	Testable	4	3	3.16s
CL [33]-B	Type III	Signature	Testable	Testable	Testable	4	3	3.21s
CL [33]-C ($\mathcal{B}(\mathsf{msg}) = 8$)	Type I	Signature	Testable	Testable	Testable	16	15	25.08s
BW [32]	Type I	Signature	Testable	Testable	Testable	1	0	36.38s
AGOT [4]	Type II	Signature	Testable	Testable	Testable	1	0	2.05s
BB [27]	Type III	Signature	N/A	Testable	Testable	2	2	2.30s
LG [49]	Type III	Signature	N/A	Testable	Testable	2	2	2.08s
ACDKNO [2]	Type III	Signature	Testable	Testable	Testable	12	20	3.20s
Dodis [35] ($\lvert C(x)\rvert = 3$)	Type I	VRF	Testable	Testable	Testable	18	30	5.29s
Dodis [35] ($\lvert C(x)\rvert = 4$)	Type I	VRF	Testable	Testable	Testable	28	49	95.5s
Lysyanskaya [50] ($\lvert C(x)\rvert = 5$)	Type III	VRF	Testable	Testable	Testable	5	4	10.38s
DY [36]	Type I	VRF	N/A	Testable	Testable	3	3	2.01s
Jager [48] ($\lvert H(x)\rvert = 4$)	Type III	VRF	Testable	Testable	Testable	5	4	5.12s
RW [54] ($a = 8$)	Type I	CP-ABE	Testable	Testable	Testable	9	8	13.96s
100-DDH	Type I	Custom	Testable	Testable	Testable	1	0	3.78s
DLIN	Type I	Custom	Unknown	Not Testable	Unknown	0	0	0.03s
Custom Testcase 1	Type I	Custom	N/A	Testable	Testable	2	2	1.89s
Custom Testcase 2	Type I	Custom	N/A	Testable	Testable	3	3	2.07s
Custom Testcase 3	Type I	Custom	N/A	Testable	Testable	2	2	1.93s
Custom Testcase 4	Type III	Custom	N/A	Testable	Testable	7	9	2.07s
Custom Testcase 5	Type III	Custom	N/A	Testable	Testable	11	17	2.11s
Custom Testcase 6	Type I	Custom	N/A	Testable	Testable	102	103	3.55s

Table 1: The output of AutoRationalPPE **on various PPE circuit testability problems. Here, ℓ represents the number of delegation levels in a HIBE scheme, $\lvert H(\mathsf{id})\rvert$ 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**, $\lvert C(x)\rvert$ denotes the length of encoding of input x, $\lvert H(x)\rvert$ denotes the length of encoding of input x, and a denotes the number of attributes. The execution time is mentioned in seconds. Here N/A denotes the fact that** AutoCircuitPPE **does not accept input with rational polynomials.**

AutoCircuitPPE, we simplified checking whether the constant d is relatively prime to $p-1$ in Rule 3a and 3b, by checking whether d is a small prime ($d \in \{1, 3, 5, 7, 11\}$), as none of the real world schemes have polynomials with high degree on their variables. Also, if a PPE is trivially True/False,[17] we replace the PPE with True/False accordingly.

We evaluated our tool various, IBE, VRF, Signature schemes and summarize our test results for 35 schemes in Table 1. For IBE schemes, we ran our tool to compute a PPE circuit which tests for well-formedness of a secret key of an identity given the master public key and the identity. For Verifiable Random Function (VRF) schemes, we aimed to construct a PPE circuit which tests for validity of VRF output and proof of pseudorandomness given the verification key and VRF input. For signature schemes, we ran the tool to output a PPE circuit which acts as a verification procedure that checks the well-formedness of a signature given message and verification key. We encoded each of the schemes into a PPE

[17] For example, if denominator polynomial h_k is a constant in Rule3a, then the circuit C is trivially True.

problem instance similar to [46] (See [46] Section 5.2 for more details). As in [46], we encode the VRF bit string input of [35, 48, 50] schemes as a vector of $\mathbb{Z}_p$ variables. 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.

We demonstrate the flexibility of our tool by testing it on problem instances in all Type I, II and III pairing settings. We note that our rules only supersede the rules proposed by AutoCircuitPPE. Consequently, AutoRationalPPE outputs a PPE testing circuit for all the problem instances on which AutoCircuitPPE outputs a PPE circuit. Additionally, AutoRationalPPE outputs PPE testing circuit for many schemes which include rational polynomials such as Boneh-Boyen IBE [25] and signatures [27], Gentry IBE [37], Le-Gabillon multisignatures [49], Dodis-Yampolskiy VRF [36] and many of other custom testcases. Even though the QSearch has exponential time complexity, it runs pretty fast on many real world schemes. After running QSearch algorithm in Section 3.3, we optimized the output circuit to remove any redundant operations. For example, if the same sub-circuit occurs in 2 different places, we compute it only once. These optimizations are adapted from AutoCircuitPPE tool. We display the number of PPE gates and Boolean gates post-optimization in Table 1.

We also tested our tool on a few custom examples containing rational polynomials, some of them having more than 100 polynomials. In the custom testcase 6 (inspired by DDHI problem), the trusted set contains polynomial $F_1 = a$ in the group $\mathbb{G}_1$, the untrusted set contains polynomials $\{F_2 = a^2, F_3 = a^3, F_4 = a^4, \cdots, F_{99} = a^{99}, F_{100} = 1/a^{100}, F_{101} = 1/a\}$ in the group $\mathbb{G}_1$. The problem can be tested using the PPEs $F_2 = e(F_1, F_1)$, $e(F_3, g) = e(F_2, F_1)$, $\cdots, e(F_{101}, F_1) = e(g, g)$, $e(F_{100}, F_{99}) = e(F_{101}, g)$. Additionally, some logic is used to consider the case where $a = 0$ and denominator of the polynomials is invalid. More details are in Appendix D.

4.4 Open Problems

This work solves a major open problem of [47] by solving the PPE Circuit testability problem for schemes with rational polynomials. We now remark on a few exciting, open problems.

First, all work on PPE automation to date [20, 46, 47] including this work focuses on perfect verification, where each element is checked individually. Some applications (such as signatures) could use a relaxed (and possibly more efficient) verification procedure where elements only need have some proper relationship to each other. We view exploring this concept of *sufficient* verification as a useful and exciting future direction. We discuss this in more detail in Appendix E.

Second, PPE automation to date [20, 46, 47] including this work focuses only on prime order groups. These groups are often highly preferred to their composite order counterparts due to both bandwidth and run time differences. Still the composite order setting is often more unwieldy due to its use of different subgroups and reliance on the property that pairing different subgroups results in an identity. Handling the constraints of these different subgroups, while properly handling identity and undefined elements, in our framework seems non-trivial.

Finally, the current implementation of our tool (as is also the case with AutoCircuitPPE [47]) outputs the first solution it finds, instead of caching several solutions and outputting the most optimal. Since PPE gates are the most costly, that is the metric on which we'd like to optimize. As one example of non-optimality, our tool's solution for the Dodis VRF [35] takes 28 PPEs, while AutoCircuitPPE found one that takes only 25 PPEs. One might consider a "deep search" option, where the tool searches all promising branches of the search space to collect a group of solutions and then outputs the solution with the smallest number of PPE gates. The technical challenge here is performing a deep search without incurring an exponential explosion in the running time.

ACKNOWLEDGMENTS

Susan Hohenberger was supported by NSF CNS-1908181, the Office of Naval Research N00014-19-1-2294, and a Packard Foundation Subaward via UT Austin. Satyanarayana Vusirikala was supported by a UT Austin Provost Fellowship, NSF CNS-1908611, and the Packard Foundation.

The authors thank Brent Waters for helpful discussions and the ACM CCS anonymous reviewers for presentation feedback.

REFERENCES

[1] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. 2012. Constant-Size Structure-Preserving Signatures: Generic Constructions and Simple Assumptions. Cryptology ePrint Archive, Report 2012/285. https://eprint.iacr.org/2012/285.

[2] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. 2012. Constant-Size Structure-Preserving Signatures: Generic Constructions and Simple Assumptions. In *ASIACRYPT*.

[3] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Takeya Tango. 2014. Converting Cryptographic Schemes from Symmetric to Asymmetric Bilinear Groups. In *Advances in Cryptology - CRYPTO*. Springer, 241–260.

[4] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Mehdi Tibouchi. 2014. Structure-Preserving Signatures from Type II Pairings. In *Advances in Cryptology - CRYPTO 2014*. 390–407.

[5] Masayuki Abe, Fumitaka Hoshino, and Miyako Ohkubo. 2016. Design in Type-I, Run in Type-III: Fast and Scalable Bilinear-Type Conversion Using Integer Programming. In *Advances in Cryptology - CRYPTO*. Springer, 387–415.

[6] Joseph A. Akinyele, Gilles Barthe, Benjamin Grégoire, Benedikt Schmidt, and Pierre-Yves Strub. 2014. Certified Synthesis of Efficient Batch Verifiers. In *IEEE 27th Computer Security Foundations Symposium*. IEEE Computer Society, 153–165.

[7] Joseph A. Akinyele, Christina Garman, and Susan Hohenberger. 2015. Automating Fast and Secure Translations from Type-I to Type-III Pairing Schemes. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1370–1381.

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

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

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

[11] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Matthew Campagna, Ernie Cohen, Benjamin Grégoire, Vitor Pereira, Bernardo Portela, Pierre-Yves Strub, and Serdar Tasiran. 2019. A Machine-Checked Proof of Security for AWS Key Management Service. In *CCS*. 63–78.

[12] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Francois Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. 2017. A Fast and Verified Software Stack for Secure Function Evaluation. In *CCS 2017*.

[13] José Bacelar Almeida, Cecile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, Francois Dupressoir, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Alley Stoughton, and Pierre-Yves Strub. 2019. Machine-Checked Proofs for Cryptographic Standards: Indifferentiability of Sponge and Secure High-Assurance Implementations of SHA-3. In *CCS*. 1607–1622.

[14] Miguel Ambrona, Gilles Barthe, Romain Gay, and Hoeteck Wee. 2017. 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*. ACM, 647–664.

[15] Miguel Ambrona, Gilles Barthe, and Benedikt Schmidt. 2016. Automated Unbounded Analysis of Cryptographic Constructions in the Generic Group Model. In *Advances in Cryptology - EUROCRYPT*. Springer, 822–851.

[16] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2019. SoK: Computer-Aided Cryptography. Cryptology ePrint Archive, Report 2019/1393. https://eprint.iacr.org/2019/1393.

[17] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. 2015. Mind the Gap: Modular Machine-Checked Proofs of One-Round Key Exchange Protocols. In *Advances in Cryptology - EUROCRYPT*. Springer, 689–718.

[18] Gilles Barthe, Francois Dupressoir, Benjamin Gregoire, Alley Stoughton, and Pierre-Yves Strub. 2018. EasyCrypt: Computer-Aided Cryptographic Proofs. https://www.easycrypt.info/trac/.

[19] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. 2014. Automated Analysis of Cryptographic Assumptions in Generic Group Models. In *Advances in Cryptology - CRYPTO*. Springer, 95–112.

[20] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt, and Mehdi Tibouchi. 2015. Strongly-Optimal Structure Preserving Signatures from Type II Pairings: Synthesis and Lower Bounds. In *Public-Key Cryptography - PKC*. 355–376.

[21] Gilles Barthe, Xiong Fan, Joshua Gancher, Benjamin Grégoire, Charlie Jacomme, and Elaine Shi. 2018. Symbolic Proofs for Lattice-Based Cryptography. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 538–555.

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

[23] Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. 2015. Automated Proofs of Pairing-Based Cryptography. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1156–1168.

[24] Bruno Blanchet. 2006. A Computationally Sound Mechanized Prover for Security Protocols. In *2006 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 140–154.

[25] Dan Boneh and Xavier Boyen. 2004. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In *Advances in Cryptology - EUROCRYPT*. Springer, 223–238.

[26] Dan Boneh and Xavier Boyen. 2004. Secure Identity Based Encryption Without Random Oracles. In *CRYPTO*. Springer, 443–459.

[27] Dan Boneh and Xavier Boyen. 2004. Short Signatures Without Random Oracles. In *EUROCRYPT*.

[28] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *Advances in Cryptology - EUROCRYPT 2005*. 440–456.

[29] Dan Boneh and Matthew K. Franklin. 2001. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology - CRYPTO*. Springer, 213–229.

[30] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT*. Springer, 514–532.

[31] Xavier Boyen and Brent Waters. 2006. Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles). In *Advances in Cryptology - CRYPTO*. Springer, 290–307.

[32] Xavier Boyen and Brent Waters. 2006. Compact Group Signatures Without Random Oracles. In *Advances in Cryptology - EUROCRYPT 2006*. 427–444.

[33] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *Advances in Cryptology - CRYPTO*. Springer, 56–72.

[34] Ran Canetti, Alley Stoughton, and Mayank Varia. 2019. EasyUC: Using EasyCrypt to Mechanize Proofs of Universally Composable Security. In *IEEE Computer Security Foundations Symposium, CSF 2019*.

[35] Yevgeniy Dodis. 2003. Efficient Construction of (Distributed) Verifiable Random Functions. In *Public Key Cryptography - PKC*. Springer, 1–17.

[36] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *Proceedings of the 8th International Conference on Theory and Practice in Public Key Cryptography (PKC'05)*.

[37] Craig Gentry. 2006. Practical Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*. Springer.

[38] Craig Gentry. 2006. Practical Identity-Based Encryption Without Random Oracles. In *Advances in Cryptology - EUROCRYPT*. Springer, 445–464.

[39] Craig Gentry and Alice Silverberg. 2002. Hierarchical ID-Based Cryptography. In *Advances in Cryptology - ASIACRYPT*. Springer, 548–566.

[40] Vipul Goyal. 2007. Reducing Trust in the PKG in Identity Based Cryptosystems. In *Advances in Cryptology - CRYPTO*. Springer, 430–447.

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

[42] Matthew Green and Susan Hohenberger. 2007. Blind Identity-Based Encryption and Simulatable Oblivious Transfer. In *Advances in Cryptology - ASIACRYPT*. Springer, 265–282.

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

[44] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. 2018. Computer-Aided Proofs for Multiparty Computation with Active Security. In *IEEE Computer Security Foundations Symposium, CSF 2018*.

[45] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. 2015. Automated Analysis and Synthesis of Authenticated Encryption Schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 84–95.

[46] Susan Hohenberger and Satyanarayana Vusirikala. 2019. Are These Pairing Elements Correct? Automated Verification and Applications. In *ACM Conference on Computer and Communications Security*.

[47] Susan Hohenberger, Satyanarayana Vusirikala, and Brent Waters. 2020. PPE Circuits: Formal Definition to Software Automation. In *ACM Conference on Computer and Communications Security*.

[48] Tibor Jager. 2015. Verifiable Random Functions from Weaker Assumptions. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC*. Springer, 121–143.

[49] Duc-Phong Le and Alban Gabillon. 2007. A New Multisignature Scheme based on Strong Diffie-Hellman Assumption. In *Conference on security in network architecture and information systems*.

[50] Anna Lysyanskaya. 2002. Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In *Advances in Cryptology - CRYPTO*. Springer, 597–612.

[51] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. 2014. Automated Analysis and Synthesis of Block-Cipher Modes of Operation. In *IEEE 27th Computer Security Foundations Symposium*. IEEE Computer Society, 140–152.

[52] Roberto Metere and Changyu Dong. 2017. Automated Cryptographic Analysis of the Pedersen Commitment Scheme. In *MMM-ACNS 2017*.

[53] David Naccache. 2005. Secure and *Practical* Identity-Based Encryption. *IACR Cryptology ePrint Archive* (2005). http://eprint.iacr.org/2005/369

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

[55] Eftychios Theodorakis and John C. Mitchell. 2018. Semantic Security Invariance under Variant Computational Assumptions. *IACR Cryptol. ePrint Arch.* 2018 (2018), 51. http://eprint.iacr.org/2018/051

[56] Brent Waters. 2005. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*. Springer, 114–127.

[57] Brent Waters. 2009. Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In *CRYPTO*. Springer, 619–636.

A PRELIMINARIES

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

A.1 Pairings

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p. 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 $e(g^a, h^b) = e(g^b, h^a) = e(g, h)^{ab}$.
(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 prime 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 4 can handle any type. We represent identity elements of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ by $\mathbf{I}_1, \mathbf{I}_2$ and $\mathbf{I}_T$ respectively.

Given pairing parameters $(p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, we extend prior definitions [43, 46] 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 $A_i, X_i \in \mathbb{G}_1, B_i, Y_i \in \mathbb{G}_2, Z \in \mathbb{G}_T, \gamma_{ij} \in \mathbb{Z}_p$. (This is the traditional definition.)
- $A \cdot \prod_{i=1}^m X_i^{\gamma_i} = 1$, where $A, X_i \in \mathbb{G}_1, \gamma_i \in \mathbb{Z}_p$.
- $A \cdot \prod_{i=1}^n Y_i^{\gamma_i} = 1$, where $A, Y_i \in \mathbb{G}_2, \gamma_i \in \mathbb{Z}_p$.

The second two PPE formats do not enable any additional functionality over the traditional definition. However, they will later help obtain more efficient identity tests. We sometimes rearrange the terms of a PPE to improve readability. We observe that under the above definition, one can employ a PPE to perform an identity test in groups $\mathbb{G}_1, \mathbb{G}_2$ or $\mathbb{G}_T$, either for a single element or according to any of the above combinations of products and exponents.

A.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.

B SHORTHAND NOTATIONS FOR CIRCUITS

For completeness, we include the shorthand notations for PPE circuits due to [47] which we also use in our presentation.

- MakeCircuit$(\mathcal{G}, m, \boldsymbol{\alpha}, P)$: Given group structure $\mathcal{G}$, number of inputs m, group identifiers $\boldsymbol{\alpha}$, and a PPE P, the function outputs a PPE circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}, \text{out}, \text{GateType}, A, B)$, where $N = 1$, Gates $= \{m+1\}$, out $= m+1$, GateType$(m+1) = (\text{PPE}, P)$, $A = \emptyset, B = \emptyset$.
- C_{acc}: We use the notation C_{acc} to denote the circuit MakeCircuit$(\mathcal{G}, m, \boldsymbol{\alpha}, P)$, where P is an always accepting PPE (for example, $g_1 = g_1$).
- Shift(C, k) : Given circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}, \text{out}, \text{GateType}, A, B)$ and integer $k \geq 1$, function Shift(C, k) outputs a circuit C' obtained by shifting the gate names Gates by an offset k i.e., $C' = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}', \text{out}', \text{GateType}', A', B')$, where Gates$' = \{g + k : g \in \text{Gates}\}$, out$' =$ out $+ k$, GateType$'(g+k) =$ GateType(g), $A'(g+k) = A(g)$ and $B'(g+k) = B(g)$, whenever $A(g), B(g)$ are defined. Note: Shift(C, k) still has $\{1, 2, \cdots, m\}$ as the input wires.
- C_1 OP C_2 (where OP $\in \{\text{AND}, \text{OR}\}$): Given circuits $C_1 = (\mathcal{G}, m, \boldsymbol{\alpha}, N_1, \text{Gates}_1, \text{out}_1, \text{GateType}_1, A_1, B_1)$ and $C_2 = (\mathcal{G}, m, \boldsymbol{\alpha}, N_2, \text{Gates}_2, \text{out}_2, \text{GateType}_2, A_2, B_2)$, let k be the smallest integer not in Gates$_1$. Let $C_2' =$ Shift$(C_2, k) = (\mathcal{G}, m, \boldsymbol{\alpha}, N_2, \text{Gates}_2', \text{out}_2', \text{GateType}_2', A_2', B_2')$. The circuit C_1 OP C_2 is given by $(\mathcal{G}, m, \boldsymbol{\alpha}, N_1 + N_2 + 1, \text{Gates}, \text{out}, \text{GateType}, A, B)$, where out is the smallest integer not in Gates$_1 \cup$ Gates$_2'$, the set Gates $=$ Gates$_1 \cup$ Gates$_2' \cup \{\text{out}\}$, the functions

$$\text{GateType}(g) = \begin{cases} \text{GateType}_1(g) & \text{if } g \in \text{Gates}_1 \\ \text{GateType}_2'(g) & \text{if } g \in \text{Gates}_2' \\ \text{OP} & \text{if } g = \text{out} \end{cases}$$

$$A(g) = \begin{cases} A_1(g) & \text{if } g \in \text{Gates}_1 \\ A_2'(g) & \text{if } g \in \text{Gates}_2' \\ \text{out}_1 & \text{if } g = \text{out} \end{cases}$$

$$B(g) = \begin{cases} B_1(g) & \text{if } g \in \text{Gates}_1 \\ B_2'(g) & \text{if } g \in \text{Gates}_2' \\ \text{out}_2' & \text{if } g = \text{out} \end{cases}$$

- NOT C : Given circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}, \text{out}, \text{GateType}, A, B)$, we let NOT C denote the circuit $(\mathcal{G}, m, \boldsymbol{\alpha}, N+1, \text{Gates}', \text{out}', \text{GateType}', A', B')$, where out$'$ is the smallest integer not in Gates, the set Gates$' =$ Gates $\cup \{\text{out}'\}$, functions

$$\text{GateType}'(g) = \begin{cases} \text{GateType}(g) & \text{if } g \in \text{Gates} \\ \text{NOT} & \text{if } g = \text{out}' \end{cases}$$

$$A(g) = \begin{cases} A(g) & \text{if } g \in \text{Gates} \\ \text{out} & \text{if } g = \text{out}' \end{cases}$$

and B' is the same as B.

C PROOFS OF CORRECTNESS

C.1 Proof of Correctness for Rule 1

Proof. We observe that every PPE challenge for Π is also a challenge for Π', as they all share the same group structure, the number of elements of m, and the group indicator vector α. Consider any testing circuit C' for Π'. We now argue by contradiction that if $C \wedge C'$ is not a testing circuit for Π, then C' cannot be a testing circuit for Π'. Since $C \wedge C'$ is not testing set for Π, then either:

- Case 1: There exists a YES challenge $\mathbf{R}$ for Π such that $C \wedge C'$ is not satisfied, or
- Case 2: There exists a NO challenge $\mathbf{R}$ for Π such that $C \wedge C'$ is satisfied.

We now analyze each of these cases.

Case 1: We know that $C \wedge C'$ is not satisfied by the YES challenge $\mathbf{R}$. By the definition of being a YES challenge, there exists a variable assignment $\mathbf{u}$ s.t. $R_i = g_{\alpha_i}^{f_i(\mathbf{u})/h_i(\mathbf{u})}$ for all i. We take this in two subcases.

Case 1(a): Suppose that $\mathbf{R}$ satisfies PPE circuit C but not the circuit C'. We know that $\mathbf{R}$ is also a YES challenge for Π' (it can use the same settings $\mathbf{u}$ for the variables), but for which C' is not satisfied. This contradicts the starting assumption that C' was a testing circuit for Π'.

Case 1(b): Suppose that $\mathbf{R}$ does not satisfy the PPE circuit C. As all the elements in $\mathbf{R}$ are well-formed, we know that $h_j(\mathbf{u}) \neq 0$ for all j. As a result, $\mathbf{u}$ satisfies eqs. (3) and (4) iff $\mathbf{u}$ satisfies eqs. (1) and (2). We know that eqs. (3) and (4) are satisfied for all the variable

assignments. This means, the variable assignment $\mathbf{u}$ and thereby the PPE challenge R satisfies eqs. (1) and (2). This means $\mathbf{R}$ does satisfy the circuit C.

Case 2: Here $\mathbf{R}$ is a NO challenge for Π but $C \wedge C'$ is satisfied. By Definition 2.2 of a NO challenge for Π, there exists an assignment to $\mathbf{u}^{\mathsf{InTrusted}}$ such that for all $i \in$ Trusted, $R_i = g_{\alpha_i}^{f_i(\mathbf{u})/h_i(\mathbf{u})}$. $\mathbf{R}$ is either a NO challenge or an INVALID challenge for Π'. We argue that $\mathbf{R}$ is also a NO challenge for Π', by showing that $\mathbf{u}^{\mathsf{InTrusted}}$ also satisfies $R_k = g_{\alpha_k}^{f_k(\mathbf{u})/h_k(\mathbf{u})}$.

This follows from the fact that PPE C is satisfied by this challenge and that C explicitly tests that R_k is computed this way, possibly with respect to an equivalent polynomial for $f_k/h_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j]$ and $h_k \neq 0$. Now since $\mathbf{R}$ is NO challenge for Π', it remains to see how it performs with respect to the circuit C'. However, since $C \wedge C'$ is satisfied by this challenge $\mathbf{R}$, then C' is satisfied as well. This contradicts the original assumption that C' was a testing circuit for Π'. ∎

C.2 Proof of Correctness for Rule 2

Proof. Consider any PPE challenge $\mathbf{R} = (R_1, R_2, \cdots R_m)$ for problem Π, and any testing circuits C', C'' for Π', Π'' respectively. We first observe that $\mathbf{R}$ is also a valid PPE challenge for Π' and Π''. This is because both share the same group structure, the number of elements m, and the group indicator vector α. We prove that if $\mathbf{R}$ is a YES challenge for Π, then it satisfies circuit Z defined above, and if $\mathbf{R}$ is a NO challenge for Π, it does not satisfy the circuit Z. We organize the proof into four cases.

Case 1 ($\mathbf{R}$ is a YES challenge for Π & IsIdentity unsatisfied):
In this case, by definition, there exists an assignment of variables $\mathbf{v}$ such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in [m]$. As each R_ℓ is a well defined group element, this means $h_\ell(\mathbf{v}) \neq 0$. This means $\mathbf{v}$ satisfies eqs. (5) and (6) iff $\mathbf{v}$ satisfies eqs. (7) and (8). We choose $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ such that eqs. (7) and (8) are satisfied for all variable assignments. Therefore, $\mathbf{v}$ satisfies eqs. (5) and (6). As $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in [m]$, the PPE challenge R also satsifies the circuit C. We also observe that $\mathbf{R}$ is a YES challenge for Π'. This is because Π and Π' have the same set of polynomials $\{f_j/h_j\}_{j\in[m]}$ and only differ in the Trusted set. As a result, $\mathbf{R}$ satisfies the circuit (NOT IsIdentity) $\wedge\, C \wedge C'$, thus satisfying Z.

Case 2 ($\mathbf{R}$ is a YES challenge for Π & satisfies IsIdentity): Let $h = (\sum_{j=1}^{|\mathbf{s}_\alpha|} b_j \cdot \mathbf{s}_\alpha[j])$. We know that $f_\ell = \mathsf{Poly1}_\ell \cdot h + \mathsf{Poly2}_\ell, h_\ell = \mathsf{Poly3}_\ell \cdot h + \mathsf{Poly4}_\ell$ for polynomials $\mathsf{Poly1}_\ell, \mathsf{Poly3}_\ell$, where $\mathsf{Poly2}_\ell/\mathsf{Poly4}_\ell$ was replaced as the ℓ^{th} polynomial in Π''. Consider any assignment of $\mathbf{v}$ s.t. $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$, $\forall \ell \in [m]$. As R_ℓ is a well-defined element, $h_\ell(\mathbf{v}) \neq 0$ and thereby $H_\alpha(\mathbf{v}) \neq 0$. We know that $\prod_j U_\alpha[j]^{b_j} = I_\alpha$ and therefore $\sum_j b_j \cdot \mathbf{s}_\alpha[j] = \sum_j b_j \cdot \hat{f}_\alpha[j]/H_\alpha$ evaluates to 0 for the variable assignment $\mathbf{v}$. This implies, $h(\mathbf{v}) = 0$.

We now break this case into 2 subcases - (2a) $\Pi'' \neq \perp$, (2b) $\Pi' = \perp$.

Case 2(a): In this case, we want to show that $\mathbf{R}$ is a YES challenge for Π''. As $h(\mathbf{v}) = 0$, $R_\ell = g_{\alpha_\ell}^{\mathsf{Poly2}_\ell(\mathbf{v})/\mathsf{Poly4}_\ell(\mathbf{v})}$ for each $\ell \in [m]$. Therefore, $\mathbf{R}$ is a YES instance for Π'' and satisfies the circuit IsIdentity $\wedge\, C''$, thus satisfying Z.

Case 2(b): We argue that this case never occurs. As $\Pi'' = \perp$, we know that there is an index j s.t. $\mathsf{Poly4}_j$ is a 0 polynomial. This means, $R_j = g_{\alpha_j}^{f_j(\mathbf{v})/h_j(\mathbf{v})} = g_{\alpha_j}^{\mathsf{Poly2}_j(\mathbf{v})/\mathsf{Poly4}_j(\mathbf{v})}$ is not a well-defined element.

Case 3 ($\mathbf{R}$ is a NO challenge for Π & IsIdentity unsatisfied):
Since we assume $\mathbf{R}$ does not satisfy the circuit IsIdentity in this case, we focus only on whether $\mathbf{R}$ satisfies $C \wedge C'$. By definition, $\mathbf{R}$ is a NO challenge for Π, and therefore it cannot be a YES challenge for Π', as both Π and Π' share the same set of polynomials. (Either it will be a NO challenge or an invalid challenge; the latter in the case where the single element difference in the Trusted set between the two problems was an improperly formed element.) Observe that if $\mathbf{R}$ satisfies C, then $\mathbf{R}$ is a NO instance for Π'. Consider any assignment of variables $\mathbf{v}$ such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. If $\mathbf{R}$ satisfies C, it means $R_k = g_{\alpha_k}^{f_k(\mathbf{v})/h_k(\mathbf{v})}$[18]. Consequently, $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for each $\ell \in$ Trusted $\cup\, \{k\}$, and Π' is a NO instance. Therefore, $\mathbf{R}$ does not simultaneously satisfy the circuits C and C' and thereby does not satisfy Z.

Case 4 ($\mathbf{R}$ is a NO challenge for Π & satisfies IsIdentity): Suppose $\Pi'' = \perp$, then $\mathbf{R}$ certainly doesn't satisfy Z. Suppose $\Pi'' \neq \perp$. In this case, we argue that $\mathbf{R}$ is a NO challenge for Π''. Let $h = (\sum_j b_j \cdot \mathbf{s}_\alpha[j])$. We know that $f_\ell = \mathsf{Poly1}_\ell \cdot h + \mathsf{Poly2}_\ell, h_\ell = \mathsf{Poly3}_\ell \cdot h + \mathsf{Poly4}_\ell$ for some polynomials $\mathsf{Poly1}_\ell, \mathsf{Poly3}_\ell$, where $\mathsf{Poly2}_\ell/\mathsf{Poly4}_\ell$ was replaced as the ℓ^{th} polynomial in Π''. Consider any assignment of variables $\mathbf{v}$ such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As $(\prod_{j=1}^{|\mathbf{s}_\alpha|} U_\alpha[j]^{b_j}) = I_\alpha$, the polynomial $(\sum_j b_j \cdot \mathbf{s}_\alpha[j])$ evaluates to 0 for the variable assignment $\mathbf{v}$. Therefore, $R_\ell = g_{\alpha_\ell}^{\mathsf{Poly2}_\ell(\mathbf{v})/\mathsf{Poly4}_\ell(\mathbf{v})}$ for each $\ell \in$ Trusted. Moreover, $\mathbf{R}$ cannot be a YES instance for Π''. This is because if there a variable assignment $\mathbf{v}$ such that $R_\ell = g_{\alpha_\ell}^{\mathsf{Poly2}_\ell(\mathbf{v})/\mathsf{Poly4}_\ell(\mathbf{v})}$ for each $\ell \in [m]$, that would mean $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for each $\ell \in [m]$ which contradicts our initial assumption that $\mathbf{R}$ is a NO instance for Π. Therefore, $\mathbf{R}$ does not satisfy the circuit C'', and thereby does not satisfy Z.

∎

C.3 Proof of Correctness for Rule 3a

Proof. Consider any PPE challenge $\mathbf{R} = (R_1, R_2, \cdots R_m)$ for problem Π, and any testing circuits C', C'' for Π', Π'' respectively. We first observe that $\mathbf{R}$ is also a valid PPE challenge for Π' and Π''. This is because both share the same group structure, the number of elements m, and the group indicator vector α. We prove that if $\mathbf{R}$ is a YES challenge for Π, then it satisfies circuit Z defined above, and if $\mathbf{R}$ is a NO challenge for Π, it does not satisfy the circuit Z. We organize the proof into four cases.

[18] Note that this crucially relies on the fact that $\prod_j U_\alpha[j]^{b_j} \neq I_\alpha$ and therefore $\sum_j b_j \cdot \mathbf{s}_\alpha[j]$ does not evaluate to 0 for the variable assignment $\mathbf{v}$.

Case 1 (**R** is a YES challenge for Π & IsIdentity unsatisfied): We first observe that **R** is also a YES challenge for Π', as Π and Π' have the same set of polynomials and only differ in the Trusted set. As a result, **R** satisfies circuit C'. As **R** is a YES instance, there exists a variable assignment **v** such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all ℓ. As R_ℓ is a well-defined element, we know that $h_\ell(\mathbf{v}) \neq 0$. As $h_k \cdot H_T \equiv \sum_{j=1}^{|\mathbf{s}_T|} b_j \cdot \hat{f}_T[j]$, we know that $\sum_{j=1}^{|\mathbf{s}_T|} b_j \cdot \hat{f}_T[j]/H_T$ does not evaluate to 0 on **v**. Therefore $\prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{b_j}$ does not evaluate to identity and **R** satsifies the circuit C, and thereby satisies the circuit (NOT IsIdentity) $\wedge\, C \wedge C'$, and thus satisfies Z.

Case 2 (**R** is a YES challenge for Π & satisfies IsIdentity): Consider any assignment of variables **v** s.t. $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$, $\forall \ell \in [m]$. As R_ℓ is a well-defined element, $h_\ell(\mathbf{v}) \neq 0$ and thereby $H_T(\mathbf{v}) \neq 0$. We know that $\prod_j U_T[j]^{a_j} = I_T$ and therefore $\sum_j a_j \cdot \mathbf{s}_T[j] = \sum_j a_j \cdot \hat{f}_T[j]/H_T$ evaluates to 0 for the variable assignment **v**. This implies, $h'(\mathbf{v}) = 0$. We know that $f_\ell = \mathsf{Poly1}_\ell \cdot h' + \mathsf{Poly2}_\ell, h_\ell = \mathsf{Poly3}_\ell \cdot h' + \mathsf{Poly4}_\ell$ for some polynomials $\mathsf{Poly1}_\ell, \mathsf{Poly3}_\ell$, where $\mathsf{Poly2}_\ell/\mathsf{Poly4}_\ell$ was replaced as the ℓ^{th} polynomial in Π''.

Suppose $\Pi'' \neq \perp$. In this case, $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})} = g_{\alpha_\ell}^{\mathsf{Poly2}_\ell(\mathbf{v})/\mathsf{Poly4}_\ell(\mathbf{v})}$ for each $\ell \in [m]$, and **R** is a YES instance for Π'' and satisfies the circuit IsIdentity $\wedge\, C''$, and thus satisfies Z.

Suppose $\Pi'' = \perp$. In this case, we know that there is an index j s.t. $\mathsf{Poly4}_j$ is a 0 polynomial. This means, $R_j = g_{\alpha_j}^{f_j(\mathbf{v})/h_j(\mathbf{v})} = g_{\alpha_j}^{\mathsf{Poly2}_j(\mathbf{v})/\mathsf{Poly4}_j(\mathbf{v})}$ is not a well-defined element. Therefore, such a case never occurs.

Case 3 (**R** is a NO challenge for Π & IsIdentity unsatisfied): In this case, **R** is not a YES instance for Π' as Π and Π' share the same set of polynomials. (Either it will be a NO challenge or an invalid challenge; the latter in the case where the single element difference in the Trusted set between the two problems was an improperly formed element.) We know that, there exists an assignment of InTrusted variables $\{v_i\}_{i \in \mathsf{InTrusted}}$ such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As **R** does not satisfy the circuit IsIdentity, $\prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} \neq \mathbf{I}_T$, which means $h'(\mathbf{v}) \neq 0$. Suppose $\{v_i\}_{i \in \mathsf{InTrusted}}$ satisfies the circuit C. Then, $h_k(\mathbf{v}) \neq 0$, and for every possible value of R_k and h'', one can solve for u_j such that $R_k = g_{\alpha_k}^{(h' \cdot u_j^d + h'')/h_k}$. This is because of our condition that d does not divide $p-1$. Consequently, there exists a variable assignment **v** such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$, $\forall \ell \in$ Trusted $\cup \{k\}$, and therefore **R** is a NO challenge for Π' and does not satisfy C'. Because it does not satisfy $C \wedge C'$, it cannot satisfy Z.

Case 4 (**R** is a NO challenge for Π & satisfies IsIdentity): Suppose $\Pi'' = \perp$, then **R** doesn't satisfy Z. Suppose $\Pi'' \neq \perp$. We argue that **R** is a NO challenge for Π''. We know that for any $\ell \in [m]$, $f_\ell = \mathsf{Poly1}_\ell \cdot h' + \mathsf{Poly2}_\ell, h_\ell = \mathsf{Poly3}_\ell \cdot h' + \mathsf{Poly4}_\ell$ for some polynomials $\mathsf{Poly1}_\ell, \mathsf{Poly3}_\ell$, where $\mathsf{Poly2}_\ell/\mathsf{Poly4}_\ell$ is the ℓ^{th} polynomial of Π''. Let $\{v_i\}_{i \in \mathsf{InTrusted}}$ be any variable assignment such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As **R** satisfies the circuit IsIdentity, $h'(\mathbf{v}) = 0$, and $R_\ell = g_{\alpha_\ell}^{\mathsf{Poly2}_\ell(\mathbf{v})/\mathsf{Poly4}_\ell(\mathbf{v})}$, $\forall \ell \in$ Trusted. Furthermore, **R** cannot be a YES instance for Π''. This is because if **R** is a YES for Π'', then there exists a variable assignment **v** such that $R_\ell = g_{\alpha_\ell}^{\mathsf{Poly2}_\ell(\mathbf{v})/\mathsf{Poly4}_\ell(\mathbf{v})} = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in [m]$, which contradicts our assumption that **R** is a NO instance for Π. Therefore, **R** is a NO challenge for Π'' and does not satisfy C'', thus it cannot satisfy Z. ∎

C.4 Proof of Correctness for Rule 3b

Proof. This proof is very similar to the proof of Rule 3a. For the sake of completeness, we present the full proof. Consider any PPE challenge $\mathbf{R} = (R_1, R_2, \cdots R_m)$ for problem Π, and any testing circuits C', C'' for Π', Π'' respectively. We first observe that **R** is also a valid PPE challenge for Π' and Π''. This is because both share the same group structure, the number of elements m, and the group indicator vector α. We prove that if **R** is a YES challenge for Π, then it satisfies circuit Z defined above, and if **R** is a NO challenge for Π, it does not satisfy the circuit Z. We organize the proof into four cases.

Case 1 (**R** is a YES challenge for Π & IsIdentity unsatisfied): We first observe that **R** is also a YES challenge for Π', as Π and Π' have the same set of polynomials and only differ in the Trusted set. As a result, **R** satisfies circuit C'. Also, for every satisfying assignment **v** that satsifies $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all ℓ, we know that either $f_k(\mathbf{v}) \neq 0$ or ($f_k(\mathbf{v}) = 0$ and $r_k = 0$). Therefore, **R** satsifies the circuit C, and thereby satisies the circuit (NOT IsIdentity) $\wedge\, C \wedge C'$, and thus satisfies Z.

Case 2 (**R** is a YES challenge for Π & satisfies IsIdentity): The proof of this case is identical to the proof of Case 2 in Rule 3a.

Case 3 (**R** is a NO challenge for Π & IsIdentity unsatisfied): In this case, **R** is not a YES instance for Π' as Π and Π' share the same set of polynomials. (Either it will be a NO challenge or an invalid challenge; the latter in the case where the single element difference in the Trusted set between the two problems was an improperly formed element.) We know that, there exists an assignment of InTrusted variables $\{v_i\}_{i \in \mathsf{InTrusted}}$ such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As **R** does not satisfy the circuit IsIdentity, $h'(\mathbf{v}) \neq 0$. Suppose $\{v_i\}_{i \in \mathsf{InTrusted}}$ satisfies the circuit C. It means either $f_k(\mathbf{v}) \neq 0$ or ($f_k(\mathbf{v}) = 0$ and $R_k = \mathbf{I}_{\alpha_k}$). In the first case, for every possible value of R_k and h'', one can solve for u_j such that $R_k = g_{\alpha_k}^{f_k/(h' \cdot u_j^d + h'')}$. This is because of our condition that d does not divide $p-1$. In the second case, any assignment of u_j along with $\{v_i\}_{i \in \mathsf{InTrusted}}$ satisfies $R_k = g_{\alpha_k}^{f_k(\mathbf{v})/h_k(\mathbf{v})}$. Consequently, there exists a variable assignment **v** such that $R_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})/h_\ell(\mathbf{v})}$, $\forall \ell \in$ Trusted $\cup \{k\}$, and therefore **R** is a NO challenge for Π' and does not satisfy C'. Because it does not satisfy $C \wedge C'$, it cannot satisfy Z.

Case 4 (**R** is a NO challenge for Π & satisfies IsIdentity): The proof of this case is identical to the proof of Case 4 in Rule 3a. ∎

Input File Example

```
maps G1 * G1 ->GT.
Zp_vars [id,r].
trusted_polys [F1 = x, F2 = y] in G1.
untrusted_polys [F3 = 1/(id + x + r*y)] in G1.
```

Figure 10: Input file for Boneh Boyen IBE scheme.

D MORE EXAMPLES

In this section, we provide more sample test cases describing how to use the AutoRationalPPE tool.

D.1 Boneh Boyen IBE

In this scheme, a central authority uses its master secret key msk to generate a secret key for a user with identity id. The user would like to verify whether the central authority gave him a well-formed key. We want to use AutoRationalPPE to verify whether the secret key for user id is well-formed. Here Trusted is the set of elements in the master public key mpk and identity id. The non-trusted elements are the elements in the secret key $\mathsf{sk}_{\mathsf{id}}$. For the sake of completeness, we first present the Boneh Boyen IBE scheme [25].

- $\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{mpk}, \mathsf{msk})$: Sample a Type-I pairing group $(\mathbb{G}_1, \mathbb{G}_T, e)$ of prime order p. Select a random group generator $g \leftarrow \mathbb{G}_1$, and random elements $x, y \leftarrow \mathbb{Z}_p^*$. Output $\mathsf{mpk} = (g, g^x, g^y)$, $\mathsf{msk} = (x, y)$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \rightarrow \mathsf{sk}_{\mathsf{id}}$: Pick a random value $r \leftarrow \mathbb{Z}_p$ and output $(r, g^{1/(\mathsf{id}+x+r*y)})$. In case $(\mathsf{id}+x+r*y) = 0 \mod p$, try again with a different r.

We now describe the input to our tool in Figure 10.

We present the output generated by our tool in Figure 11. Note that the trusted set contains 2 elements (id, r) in $\mathbb{Z}_p$. We input these elements using `Zp_vars[id,r].` syntax. We describe the gates of the final circuit at the end using **Gxx** notation. For PPE gates, we mention the PPE computed by the gate. For boolean gates, we describe the boolean logic performed by the gate.

D.2 Custom Example

In this section, we describe our custom test case 5 mentioned in Table 1. We design the test case so that all the four rules are invoked here. We describe the input to our tool in Figure 12, and then describe the output generated by our tool in Figures 13 to 15. For each recursive invocation of the QSearch algorithm, the output file describes the set of trusted and untrusted polynomials and the rule that is applied to the problem. The gates of the final optimized circuit are described at the end in Figure 15 using **Gxx** notation. For PPE gates, we mention the PPE computed by the gate. For boolean gates, we mention the boolean logic performed by the gate.

E ON PERFECT VERSUS SUFFICIENT VERIFICATION

Recall that the goal of our tool is to verify U with respect to T. As in [46, 47], we define a PPE Circuit that must achieve *perfect verification* where it outputs one if and only if all input elements of U are exactly as specified. That is, if some element F is supposed to have the form g^{a+b}, then it must. Perfect verification gives total

Output File Example

```
F0 = 1 in G1      F0 = 1 in GT      F1 = x in G1
F2 = y in G1      F3 = 1/id + x + r*y in G1
F0^id = id in G1      F0^r = r in G1      F0^id = id in GT
F0^r = r in GT      F1^id = id*x in G1      F1^r = r*x in G1
F2^id = id*y in G1      F2^r = r*y in G1

Trusted in G1: F1 = x, F2 = y, F0^id = id, F0^r = r, F1^id = id*x, F1^r = r*x,
F2^id = id*y, F2^r = r*y,
Trusted in GT: F0^id = id, F0^r = r,
Untrusted in G1: F3 = 1/id + x + r*y,
Rule 2 applied on F3 = 1/id + x + r*y. isidentity := F2^r*F0^id*F1 = I C :=
(e(F3,F2^r*F0^id*F1) = F0 AND (NOT F2^r*F0^id*F1 = I))

Execution time : 11.597542s
The circuit output by QSearch:
(((NOT F2^r*F0^id*F1 = I) AND (e(F3,F2^r*F0^id*F1) = F0 AND (NOT
F2^r*F0^id*F1 = I))) AND ACC)

Optimized Circuit:
G1 : F2^r*F0^id*F1 = I      G2 : e(F3,F2^r*F0^id*F1) = F0
G3 : NOT G1      G4 : G3 AND G2
```

Figure 11: Output file for Boneh Boyen IBE scheme.

Input File Example

```
maps G1 * G2 ->GT.
trusted_polys [F1 = a, F2 = b] in G1.
trusted_polys [F3 = b] in G2.
untrusted_polys [F4 = a*c] in G1.
untrusted_polys [F5 = a*b, F6 = c, F7 = d, F8 = d/(b + a), F9 = x, F10 = y,
F11 = (x*a + y*b), F12 = 1/(s+a)] in G2.
```

Figure 12: Input file for the custom example.

confidence that the untrusted elements are as they should be. It helps applications like accountable authority IBE [40, 41] or oblivious transfer from blind IBE [42], where a malicious authority with a master secret key might be trying to fool a user. In some cases, the perfect verification is necessary, and thus, it is good that we can achieve it.

However, not all applications require perfect verification: signatures are a prime example. The verification equation for many signature schemes in the literature (e.g., see Section 5 of [2]) does not guarantee that the purported signature comes from the space of signatures output by the signing algorithm. Instead, in some cases, it is enough to argue that even if the purported signature is outside of this space, an adversary could not have computed it without knowledge of the secret key. (Some of these schemes verify a signature with, say, five elements using a single PPE. Thus, they do not verify that each element of the signature is correct, but rather that all five elements – whatever they are – have the proper relationship to each other.) We might call this concept *sufficient verification*, where the requirements on the PPE Circuit are relaxed to output one if and only if the elements of U have some special relationship that can be verified using T. If sufficient verification is enough for an application, it is likely to offer better efficiency. Now that this work establishes how to automate perfect verification for large classes of prime-order pairing-based systems, we view

studying how to model, search for and apply sufficient verification as a rich area for future research.

Output File Example

F0 = 1 in G1 F0 = 1 in G2 F0 = 1 in GT F1 = a in G1
F2 = b in G1 F3 = b in G2 F4 = a*c in G1
F5 = a*b in G2 F6 = c in G2 F7 = d in G2
F8 = d/a + b in G2 F9 = x in G2 F10 = y in G2
F11 = a*x + b*y in G2 F12 = 1/a + s in G2
Trusted in G1: F1 = a, F2 = b
Trusted in G2: F3 = b
Untrusted in G1: F4 = a*c, F5 = a*b, F6 = c, F7 = d, F8 = d/a + b, F9 = x, F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 1 applied to F5 = a*b/1. C := e(F5,F0) = e(F1,F3)

Trusted in G1: F1 = a, F2 = b
Trusted in G2: F3 = b, F5 = a*b
Untrusted in G1: F4 = a*c, F6 = c, F7 = d, F8 = d/a + b, F9 = x, F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 3a applied on F4 = a*c/1 and variable c. isidentity := e(F1,F0) = I, C := (NOT REJ)

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b,
Untrusted in G1: F6 = c, F7 = d, F8 = d/a + b, F9 = x, F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 2 applied on F6 = c/1. isidentity := F1 = I C := e(F6,F1) = e(F4,F0)

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b, F6 = c
Untrusted in G1: F7 = d, F8 = d/a + b, F9 = x, F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 3a applied on F7 = d/1 and variable d. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b, F6 = c, F7 = d,
Untrusted in G1: F8 = d/a + b, F9 = x, F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 2 applied on F8 = d/a + b. isidentity := F2*F1 = I C := (e(F8,F2*F1) = e(F0,F7) AND (NOT F2*F1 = I))

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b, F6 = c, F7 = d, F8 = d/a + b,
Untrusted in G1: F9 = x, F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 3a applied on F9 = x/1 and variable x. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b, F6 = c, F7 = d, F8 = d/a + b, F9 = x,
Untrusted in G1: F10 = y, F11 = a*x + b*y, F12 = 1/a + s,
Rule 3a applied on F10 = y/1 and variable y. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b, F6 = c, F7 = d, F8 = d/a + b, F9 = x, F10 = y,
Untrusted in G1: F11 = a*x + b*y, F12 = 1/a + s,
Rule 1 applied to F11 = a*x + b*y/1. C := e(F11,F0) = e(F2,F10)*e(F1,F9)

Figure 13: Output file for the custom example.

Output File Example

Trusted in G1: F1 = a, F2 = b, F4 = a*c
Trusted in G2: F3 = b, F5 = a*b, F6 = c, F7 = d, F8 = d/a + b, F9 = x, F10 = y, F11 = a*x + b*y,
Untrusted in G1: F12 = 1/a + s,
Rule 3b applied on F12 = 1/a + s and variable s. isidentity := REJ, C := ACC

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0,
Untrusted in G1: F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F6 = c/1 and variable c. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c,
Untrusted in G1: F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F7 = d/1 and variable d. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d,
Untrusted in G1: F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 2 applied on F8 = d/b. isidentity := F2 = I C := (e(F8,F2) = e(F0,F7) AND (NOT F2 = I))

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b,
Untrusted in G1: F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F9 = x/1 and variable x. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x,
Untrusted in G1: F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F10 = y/1 and variable y. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y,
Untrusted in G1: F11 = b*y, F12 = 1/s,
Rule 1 applied to F11 = b*y/1. C := e(F11,F0) = e(F2,F10)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y,
Untrusted in G1: F12 = 1/s,
Rule 3b applied on F12 = 1/s and variable s. isidentity := REJ, C := ACC

Trusted in G1: F1 = 0, F2 = b,
Trusted in G2: F3 = b, F5 = 0,
Untrusted in G1: F4 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 1 applied to F4 = 0/1. C := F4 = I

Figure 14: Output file for the custom example (Cont'd).

Output File Example

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0,
Untrusted in G1: F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F6 = c/1 and variable c. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c,
Untrusted in G1: F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F7 = d/1 and variable d. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d,
Untrusted in G1: F8 = d/b, F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 2 applied on F8 = d/b. isidentity := F2 = I C := (e(F8,F2) = e(F0,F7) AND (NOT F2 = I))

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b,
Untrusted in G1: F9 = x, F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F9 = x/1 and variable x. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x,
Untrusted in G1: F10 = y, F11 = b*y, F12 = 1/s,
Rule 3a applied on F10 = y/1 and variable y. isidentity := REJ, C := (NOT REJ)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y,
Untrusted in G1: F11 = b*y, F12 = 1/s,
Rule 1 applied to F11 = b*y/1. C := e(F11,F0) = e(F2,F10)

Trusted in G1: F1 = 0, F2 = b, F4 = 0,
Trusted in G2: F3 = b, F5 = 0, F6 = c, F7 = d, F8 = d/b, F9 = x, F10 = y, F11 = b*y,
Untrusted in G1: F12 = 1/s,
Rule 3b applied on F12 = 1/s and variable s. isidentity := REJ, C := ACC

Optimized Circuit:
G1 : e(F5,F0) = e(F1,F3) **G2** : e(F1,F0) = I **G3** : F1 = I
G4 : e(F6,F1) = e(F4,F0) **G5** : F2*F1 = I **G6** : e(F8,F2*F1) = e(F0,F7)
G7 : e(F11,F0) = e(F2,F10)*e(F1,F9) **G8** : F2 = I
G9 : e(F8,F2) = e(F0,F7) **G10** : e(F11,F0) = e(F2,F10) **G11** : F4 = I
G12 : NOT G2 **G13** : NOT G3 **G14** : G13 AND G4 **G15** : NOT G5
G16 : G15 AND G6 **G17** : G16 AND G7 **G18** : G14 AND G17
G19 : NOT G8 **G20** : G19 AND G9 **G21** : G20 AND G10
G22 : G3 AND G21 **G23** : G18 OR G22 **G24** : G12 AND G23
G25 : G11 AND G21 **G26** : G2 AND G25 **G27** : G24 OR G26
G28 : G1 AND G27

Figure 15: Output file for the custom example (Cont'd).