PPE Circuits: Formal Definition to Software Automation

Susan Hohenberger
susan@cs.jhu.edu
Johns Hopkins University

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

Brent Waters
bwaters@cs.utexas.edu
University of Texas at Austin and NTT Research

ABSTRACT

Pairing-based cryptography is widely used for its efficiency and functionality. When designing pairing-based schemes, one common task is to devise algorithms for verifying a set of untrusted group elements with respect to a set of trusted group elements. One might be searching for a verification algorithm for a signature scheme or a method for verifying an IBE/ABE private key with respect to the IBE/ABE public parameters. In ACM CCS 2019 [45], the AutoPPE software tool was introduced for automatically generating a *set* of pairing product equations (PPEs) that can verify the correctness of a set of pairing group elements with respect to a set of trusted group elements. This task is non-trivial. Some schemes (e.g., those based on dual system encryption) provably do not support any efficient algorithm for verifying the private keys with respect to the public parameters. Other schemes (e.g., the Boyen-Waters anonymous IBE) were left in a gray area by [45] – no conjunction of PPEs was known for testing them, but no proof of untestability either.

In this work, we significantly generalize and expand on the foundation of [45]. Specifically, we consider a larger space of verification algorithms, which we call *PPE Circuits*, to verify a set of untrusted group elements with respect to a set of trusted group elements. Informally, a PPE Circuit supports AND, OR, NOT and PPE gates, thus capturing all of the capability of AutoPPE while novelly enabling the verification algorithm to include arbitrary logic (as opposed to only conjunctions of PPEs). Our contributions include a formalization of PPE circuits, a provably-correct algorithm for searching for a PPE circuit given a description of the trusted and untrusted elements to be verified, and a new open-source software tool[1] called AutoCircuitPPE that realizes this algorithm. AutoCircuitPPE was tested on a host of test cases and it output PPE circuits for all "gray area" schemes left unresolved in [45] as well as several new test cases, usually in 100 seconds or less.

CCS CONCEPTS

• **Security and privacy → Logic and verification**.

KEYWORDS

Automated Proofs; Provable Security; Pairing-based Cryptography

1 INTRODUCTION

Cryptography is a powerful tool for securing digital systems, but its design and security analyses are often both complex and tedious, where a single error can cause catastrophic failure. This is an ideal situation for employing computers to help humans improve the speed, accuracy, and design of cryptographic implementations. Indeed, there has already been significant success in this area. A growing suite of software tools, e.g., [6–8, 16–18, 20, 20–22, 25, 45], have demonstrated that many cryptographic tasks can be greatly improved and/or simplified with computer aid. These tasks typically fall into one of three categories: (1) designing a scheme, (2) generating a security proof, or (3) verifying security proof. In this work, we focus on building an automated tool that helps with the first task - designing a scheme.

Like many prior works, we focus on the popular pairing-based algebraic setting. 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 map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, such that for all $g \in G_1$, $h \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, h^b) = e(g, h)^{ab}$. Following [42], a *pairing product equation* (PPE) over variables $Z, \{X_i\}_{i=1}^m, \{Y_i\}_{i=1}^n$ is an equation of the form

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

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

Pairing Element Verification is Useful but Non-Trivial. When designing pairing-based schemes, one common task is that of figuring out how to *verify* one or more group elements with respect to another set of elements using the group operations and the pairing function. One example is verifying a signature with respect to a message and public key. Another example is verifying an IBE private key with respect to the identity and public parameters. This is useful when designing new structure-preserving signature schemes [1], accountable authority IBE [39, 40] or oblivious transfer from blind IBE [41].

Surprisingly, not all IBE schemes have private keys that can be verified from the identity and public parameters [45]; making this problem non-trivial. The Waters dual system IBE [53] is one such example [45].[2]

Our Results in the Context of Prior Work. The goal of this work (and past works [19, 45]) is realizing a tool that on input a description of trusted and untrusted elements outputs either (1) an algorithm for verifying the untrusted elements with respect to the trusted elements or (2) a message unknown (meaning that the tool failed to find a verification algorithm.) The correctness requirement on any verification algorithm output by the tool is that when given properly distributed

[1] Available at https://github.com/JHUISI/auto-tools

[2] In the proof of [53], there are real and semi-functional private keys, with no overlap between these key spaces. The proof uses the fact that, under standard complexity assumptions, no efficient adversary can distinguish between a real or a semi-functional private key. Thus, the existence of a polynomial-time verification algorithm for real private keys would contradict this proof.

Confidential Submission to ACM CCS 2020, Due 20 Jan 2020, Orlando, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

trusted elements and a set of untrusted elements, it must output 1 if the untrusted elements are properly distributed and 0 otherwise. (There are no requirements on the tool when the *trusted* parameters are incorrectly distributed. E.g., when the public parameters are not honestly generated.)

To date, tools of this form are comprised of one or more rules for when an untrusted element can be moved into the trusted set. Our tool uses a set of four logic rules (see Section 4). Logic similar to Rule 1 first appeared in [19], while that of Rule 2 was introduced in [45]. Rules 3 and 4 are novel to this work and we prove their correctness in Section 4. As we will see, these new rules allow us to generate verification algorithms that perform arbitrary logic over PPEs, as opposed to only conjunctions of PPEs considered in [19, 45]. This expressiveness, in turn, allows the tool to output multiple solutions on which prior tools output unknown. We now describe the first two rules in the context of prior work and then present the new ones.

In 2015, Barthe, Fagerholm, Fiore, Scedrov, Schmidt and Tibouchi [19] built an automated tool to design optimal structure-preserving signatures in Type II[3] pairing groups. Their tool generates thousands of candidate public key and signature pairs, and then for each pair searches for a corresponding verification algorithm expressed as the conjunction of PPEs. They weed out insecure schemes using the GGA tool [18]. Their searching algorithm uses a logic similar to our Rule 1 in Section 4, which checks if an untrusted element can be verified using one PPE and pairing only with a fixed generator. E.g., an untrusted element $F \in \mathbb{G}_1$ can be moved to the trusted set, if $e(F, g_2) = A$ for some A that can be computed solely from the trusted elements and g_2 is the generator for $\mathbb{G}_2$.

Generalizing this approach beyond signatures, the AutoPPE software tool [45] takes in a description of any set of trusted and untrusted elements and outputs either a verification algorithm that consists of a *set* of pairing product equations (PPEs) or the message unknown. This tool executes quickly and worked well for many signature, VRF, and IBE test cases. It uses logic similar to our Rule 1 and Rule 2. Rule 2 in Section 4 (as an oversimplification) moves an untrusted element $F = g^u$ into the trusted set if the variable u does not yet appear in any element in the trusted set.

AutoPPE was not able to produce verification algorithms for the Boyen-Waters anonymous IBE [30], the Bellare-Kiltz-Peikert-Waters IBE [24] or the Dodis verifiable random function [34]. This was very curious and unsatisfactory. While it was *possible* that the IBE schemes might not have testable private keys (although we later discovered that they do[4]), Dodis [34] provides an algorithm for verifying his VRF proofs, so why couldn't AutoPPE find this algorithm? The answer is: it needed broader rules and support for arbitrary logic.

In particular, prior works [19, 45] allow an element F to be moved to trusted if it can be paired with a generator (in $\mathbb{G}_1$ or $\mathbb{G}_2$) and tested against an element in $A \in \mathbb{G}_T$ computable from the trusted set, e.g., $e(F, g_2) = A$. Whereas this work will allow an element to be paired with a second element B computable from the trusted set, e.g., $e(F, B) = A$, if $B \neq 1$ (and this *if* introduces the need for broader logic). Intuitively, schemes that require this more general pairing test, such as [24, 30, 34], are one class of schemes for which the new tool is an improvement.

PPE Circuits. In this work, we broaden the search rules and develop a tool that can automatically find verification algorithms that support arbitrary logic (in the form of AND, OR and NOT gates) over PPEs, solving an open problem from [45]. The new tool called AutoCircuitPPE automatically searches for a verification algorithm expressed as a *PPE Circuit* (see Section 3 for a formalization and Figure 11 for a picture example). Informally, a PPE Circuit can be thought of as a circuit with AND, OR and NOT gates where some inputs to these gates come from the evaluation of certain PPEs. The requirement on any PPE Circuit output by the tool is that given properly distributed trusted elements and a set of untrusted elements, it must output 1 if the untrusted elements are properly distributed and 0 otherwise. (That is, we require perfect correctness.) The search space of PPE Circuits is much larger than that of only the conjunction of PPEs. This makes this problem both more interesting and more challenging. But the effort was worth it: as described in Section 5, AutoCircuitPPE can find verification algorithms for the Boyen-Waters anonymous IBE [30], the Bellare et al. IBE [24], the Dodis VRF [34] and custom test cases – on which AutoPPE output unknown. This demonstrates the power of supporting arbitrary logic.

Building an Automated Tool for PPE Circuits. The heart of AutoCircuitPPE is a recursive searching algorithm that, at every step, tries to move an untrusted element into the trusted set and (possibly) adds some logic and/or PPE to the verification algorithm (PPE Circuit). At each step, it checks if any of our four logic rules apply. If at any point, no rules are applicable, then the tool outputs unknown. Once all elements are trusted, it outputs the PPE Circuit.

We already covered Rules 1 and 2. Informally, Rule 3 is a generalization of Rule 1, where an untrusted element $F \in \mathbb{G}_1$ can be moved to the trusted set, if $e(F, B) = A$ for some $A \in \mathbb{G}_T$ and $B \in G_2$ that can be computed solely from the trusted elements. However, allowing B instead of only g_2 creates an issue that must be handled carefully. What if B evaluates to g_2^0? The PPE $e(F, B) = A$ might then hold regardless of the value of F. To deal with this, our verification algorithm must be able to test if $B = g_2^0$. If the answer is no, then the PPE $e(F, B) = A$ can be used to verify F. If not, then it cannot, but one can substitute $g_2^0 = 1$ for B anywhere that it appears and then continue searching for a way to verify the simplified instance. This rule is necessary for verifying the [24, 30, 34] test cases.

Similarly, Rule 4 is a generalization of Rule 2, where (we are oversimplifying here) an element F can be moved to trusted if $F = g_1^{h_1 \cdot u + h_2}$, where h_2 may not yet be computable using elements from the trusted set, but h_1 is computable from the trusted set, and u is not yet used in any element in the trusted set. We must deal with the issue when $h_1 = 0$ and the verification algorithm must test for this and branch in its logic accordingly. All of the [24, 30, 34] test cases also required this rule.

[3] See Section 2.1; in the Type II setting, there exists an efficient isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ or from $\mathbb{G}_2$ to $\mathbb{G}_1$ but not both.

[4] We had to use our tool to find these answers. We were not aware of any known algorithms for verifying the private keys of [24, 30] and we were not able to find one by hand for the Boyen-Waters IBE [30].

In Section 5, we discuss the 29 test cases we explored for AutoCircuitPPE. AutoCircuitPPE on Boyen-Waters [30] required all four rules to output a PPE Circuit with 27 PPEs and 124 boolean gates. **To the best of our knowledge, this is the first time any verification algorithm for verifying Boyen-Waters anonymous IBE private keys with respect to the identity and public parameters has been discovered.** The software took less than 19 seconds to find this verification algorithm, which we were unable (after hours of trying) to find by hand. The most costly verification algorithm to uncover automatically was that for the Lysyanskaya VRF [47] requiring almost 110 seconds. There is provably no PPE Circuit for the Waters dual system IBE [53]; on this input, AutoCircuitPPE ran for 105 seconds before giving up and outputting unknown. Our test cases showed that this new tool is demonstrably more comprehensive in its coverage than prior tools, while still efficient enough for easy use.

1.1 Related Work

Our work builds on an impressive collection of prior work in computer automation for cryptography. We highlight a selection here. For more, we refer the reader to a recent survey of computer-aided cryptography by Barbosa et al. [15].

We previously discussed the progress of Barthe et al. [19] and Hohenberger-Vusirikala [45] from which we build upon. Additionally, we use the Generic Group Analyzer (GGA) tool of Barthe et al. [18], which analyzes cryptographic assumptions in the generic group model, and was extended to handle unbounded assumptions by Ambrona et al. [14].

Other works in computer-aided cryptographic design include: AutoBatch [8, 9] (for batching the verification of PPEs), AutoStrong [7] (for compiling a signature scheme secure under the standard definition into one that is strongly secure), AutoGroup+ [6, 7] (for translating a Type-I pairing scheme into a Type-III pairing scheme; IPConv [2, 4] is an alternative method, although not open source at this time). Ambrona et al. [13] showed how to apply computer-aided reasoning to the design of attribute-based encryption systems. In the private key setting, there are interesting automation results for blockciphers [48] and authenticated encryption [44].

There are many great tools for automating proof generation or verification, such as Cryptoverif [25], CertiCrypt [21], EasyCrypt [17] and AutoG&P [22]. In 2019, researchers impressively showed how to use EasyCrypt to machine-check a security proof for the domain management protocol of Amazon Web Services' KMS (Key Management Service) [10] and to verify cryptographic standards such as SHA-3 [12]. These tools were also used to verify proofs for key exchange protocols [16, 33], MPC protocols [43], commitment schemes [49], software stacks [11] and protocols in the UC framework [33]. Barthe et al. [22] provided a tool that translates the proofs output by AutoG&P into a format verifiable by EasyCrypt and similarly Akinyele et al. [5] showed that the proofs output by AutoBatch can be automatically verified by EasyCrypt. The AutoLWE tool [20] semi-automatically proves the security of cryptographic constructions based on Learning with Errors.

2 PRELIMINARIES

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

2.1 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 5 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 [42, 45] 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, but they will later be useful for obtaining more efficient identity tests. We sometimes rearrange the terms of a PPE to improve readability. As we will use it later, we observe that under the above definition, a PPE can be employed as 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.

3 DEFINING PPE CIRCUITS

We introduce and formally define PPE circuits. We begin with the notion of a PPE problem instance [45].

Definition 3.1 (PPE Problem Instance [45]). 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 polynomials $\mathbf{f} = (f_1, \ldots, f_m)$ over n variables in $\mathbb{Z}_p$ denoted $\mathbf{u} = (u_1, \ldots, u_n)$,
- a sequence of pairing group identifiers in $\mathcal{I} = \{1, 2, T\}$ denoted $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$,
- a set 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) = $\cup_{i \in \text{Trusted}}$\{variables used in f_i\} $\subseteq \mathbf{u}$. We simplify the notation and use InTrusted whenever the problem instance Π is implicit.

Following [45], we map cryptographic schemes into PPE problem instances by rewriting the scheme using a single group generator, when possible. For example, let g_1, g_2, g_T be group generators of groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively. Let a group element in the scheme be $h^x \cdot g_2^x \cdot g_2^y$. We rewrite this element as g_2^{xz+x+y}, by representing $h = g_2^z$ for a fresh variable z. Consequently, each group element in the scheme could be represented by their group indicator ($\mathbb{G}_1/\mathbb{G}_2/\mathbb{G}_T$) along with the polynomial present in the exponent.

Definition 3.2 (PPE Challenge [45]). Let $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}$, Trusted) be a PPE problem instance as in Definition 3.1. Let $\mathbf{F} = (F_1, \ldots, F_m)$ be comprised of pairing group elements, where each F_i is in group $\mathbb{G}_{\alpha_i}$. $\mathbf{F}$ is called a *challenge* to PPE instance Π. Challenges are classified as:

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

Following [45], we can view a 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), and therefore, we ignore this case since verification requires correctness of the elements we trust.

The goal of our work will be to (automatically) devise *circuits* that take as input a *PPE challenge* (recall Definition 3.2) and output 1 for all YES challenges and 0 for all NO challenges. That is, where prior work [45] allowed only the conjunction of PPEs to test the well-formedness of the untrusted elements; we will now combine the power of PPEs with arbitrary logic. Informally, the PPE circuit takes m group elements as input and outputs a single bit. Like regular circuits, each gate of the circuit could be an AND/OR/NOT gate. In addition, we also allow the circuit to have PPE gates. Each PPE gate has some PPE P (over formal variables $F_1, F_2, \cdots, F_m$ denoting the m input wires of the PPE circuit) hardcoded in it and outputs a boolean value representing whether the m input group elements satisfy P. Informally, in order to evaluate a PPE circuit on given m group elements $(x_1, x_2, \cdots x_m)$, we first evaluate each PPE gate on the given input (check whether the PPE is satisfied by substituting $F_i = x_i \forall i \in [m]$), and then evaluate the boolean circuit logic to obtain the final output. As observed in Section 2.1, PPEs can also capture identity tests as well as be a hardwire for 0 (the equation that is never satisfied) or 1 (the equation that is always satisfied).

We now establish some formal notation for our specialized PPE circuits adapting the more general circuit notation of Bellare, Hoang, and Rogaway [23] and Garg, Gentry, Halevi, Sahai and Waters [36].

Definition 3.3 (PPE Circuit). A *PPE circuit* C is a tuple ($\mathcal{G}, m, \boldsymbol{\alpha}$, N, Gates, out, 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, β), AND, OR, NOT\} is a function that identifies the gate functionality. In case of PPE gates, the description includes a circuit β with m Inputs wires whose logic forms that of a PPE over variables $F_1, \ldots, F_m$ where each $F_i \in \mathbb{G}_{\alpha_i}$ 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.
- A : Gates $\rightarrow$ Wires and B : Gates $\rightarrow$ 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.

This describes a circuit taking as input m group elements and outputting a single output on wire out. We now describe how to evaluate the above circuit.

Definition 3.4 (PPE Circuit Evaluation). A *PPE circuit evaluation algorithm* Eval : $C \times (x_1, \ldots, x_m)$ takes as input a PPE circuit C = ($\mathcal{G}, m, \boldsymbol{\alpha}, N$, Gates, out, GateType, A, B) together

[5] Unlike the definition of [45], we do not include the set Fixed in the PPE Problem Instance definition as we implicitly define this set of variables (called InTrusted) as those that appear in elements corresponding to the Trusted set.

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 $\{0, 1\}$.

Here we describe a "canonical" evaluation algorithm. 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 GateType(g):

- if (PPE, β), then evaluate the PPE β using the assignment to variables in $(F_1, \ldots, F_k)$. 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.

The above conditions guarantee that the circuit does not have any loops. While we chose to have AND, OR and NOT gates, this is somewhat arbitrary. We could have chosen only NAND or allowed gates with larger fan-in, etc. We abuse notation and let $C(\mathbf{x})$ denote Eval$(C, \mathbf{x})$ i.e., evaluation of the circuit C on input $\mathbf{x}$.

We next extend the notion of PPE testability and testing sets [45] to apply to PPE circuits.

Definition 3.5 (PPE Circuit Testable and Testing Circuits). A PPE problem instance $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ is said to be *PPE circuit testable* if and only if there exists a PPE circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$. 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.

3.1 A Few Shorthand Notations for Circuits

Useful shorthand is informally defined here with formalisms in Appendix A. We use MakeCircuit$(\mathbb{G}, m, \boldsymbol{\alpha}, P)$ to define a PPE circuit that computes the output of a PPE P and use C_{acc} to denote the circuit which always outputs 1.

Consider any two PPE circuits C_1 and C_2 with the same group structure $\mathcal{G}$, number of inputs m and group identifiers $\boldsymbol{\alpha}$. When building our circuits, we will use shorthand notation like (C_1 AND C_2), (C_1 OR C_2) or (NOT C_1). Informally, we use C_1 AND C_2 to denote the circuit obtained by ANDing the output wires of C_1 and C_2 (i.e., connecting output wires of C_1 and C_2 as inputs to a fresh AND gate, and considering the output wire of the fresh AND gate as output wire of the entire circuit). Similarly, we use the notation C_1 OR C_2 to denote the circuit obtained by ORing the output wires of C_1 and C_2. We use NOT C_1 to denote the circuit obtained by connecting the output wire of C_1 to a fresh NOT gate and then considering the output wire of the NOT gate to be the output wire of the entire circuit. Furthermore, when the circuits share common inputs (e.g., same PPE challenge elements), we will make sure the final circuit has only the appropriate number of input wires.

4 SEARCHING FOR A PPE TESTING CIRCUIT

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

The algorithm proceeds in a sequence of steps, wherein each step it (attempts to) "reduce the complexity" of its input, by adding a polynomial f_i to the set Trusted. So far, this is similar to AutoPPE [45], however, here we must expand the number and type of rules for when a polynomial can be moved to Trusted. In the end, if we can obtain Trusted $= [1, m]$, then we will have found a testing circuit. If at any point, Trusted $\neq [1, m]$ but none of the movement rules can be applied, the algorithm terminates and outputs unknown.

4.1 Review on Computing Completion Lists for a List of Polynomials

Our rules will make use of completion lists in the pairing setting as described by Barthe et al. [18]. Consider any list $\mathbf{f} = [f_1, \ldots, f_k]$ of polynomials along with a sequence of identifiers $\alpha_1, \cdots, \alpha_k$, where $\alpha_i \in \mathcal{I} = \{1, 2, T\}$ for all $i \leq k$. For any $i \in \mathcal{I}$, let $\mathbf{t}_i = \{f_j : \alpha_j = i\}$. We now recall the notion of completion $\mathsf{CL}(\mathbf{f}) = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\}$ of the list $\mathbf{f}$ of polynomials with respect to a group setting [18]. Intuitively, $\mathsf{CL}(\mathbf{f})$ is the list of all polynomials that can be computed by an adversary by applying pairing and isomorphism operations, when he has access to the elements in group $\mathbb{G}_i$ corresponding to the polynomials in $\mathbf{t}_i$ for $i \in I$.

Reception List

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

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

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

We now describe an algorithm to compute the completion $\mathsf{CL}(\mathbf{f})$, which is taken from [18] and handles pairing groups. The algorithm proceeds in two steps. In the first step, it computes the reception lists $\{\mathbf{r}_i\}_{i \in \mathcal{I}}$. The elements of the reception lists are monomials over variables $w_{i,j}$ for $i \in \mathcal{I}, j \in |\mathbf{t}_i|$ and are computed as shown in Figure 1.

The monomials characterize which products of elements in $\mathbf{t}$ the adversary can compute by applying pairing operations. The result of the first step is independent of the elements in the lists $\mathbf{t}$ and only depends on the lengths of the lists. In the second step, it computes the actual polynomials from the reception lists as $\mathbf{s}_i = [m_1(\mathbf{t}), \ldots, m_{|\mathbf{r}_i|}(\mathbf{t})]$ for $[m_1, \ldots, m_{|\mathbf{r}_i|}] = \mathbf{r}_i$, where every m_k is a monomial over the variables $w_{i,j}$ and $m_k(\mathbf{t})$ denotes the result of evaluating the monomial m_k by substituting $w_{i,j}$ with $\mathbf{t}_i[j]$ for $i \in \mathcal{I}$ and $j \in |\mathbf{t}_i|$.

Description of Rule 1

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

(1) If $k \in$ Trusted or $f_k \in \mathbf{f}$ has variables not in InTrusted, then abort and output $\perp$.
(2) Let the formal variables $F_1, F_2, \cdots, F_m$ 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) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{f}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \leq |\mathbf{s}_i|$, let $S_i[j] = g_{\alpha_i}^{s_i[j]}$, and let $U_i[j]$ be the pairing product term computing $S_i[j]$ in terms of formal variables $F_1, \cdots, F_m$.
(4) If there exists a constant vector $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{s}_{\alpha_k}|})$ with entries in $\mathbb{Z}_p$ such that $f_k \equiv \sum_{j=1}^{|\mathbf{s}_{\alpha_k}|} a_j \cdot \mathbf{s}_{\alpha_k}[j]$, then set the PPE
$$A := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} = F_k$$
(5) If $\alpha_k \in \{1, 2\}$, A is not set, and there exists a constant vector $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{s}_T|})$ with entries in $\mathbb{Z}_p$ such that $f_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j]$, then create the PPE
$$A := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} = \begin{cases} e(F_k, g_2) & \text{if } \alpha_k = 1 \\ e(g_1, F_k) & \text{if } \alpha_k = 2 \end{cases}$$
(6) If A is set, output the PPE circuit $C = \mathsf{MakeCircuit}(\mathcal{G}, m, \boldsymbol{\alpha}, A)$ and the PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$, else output $\perp$.
Note that for any $i \in \mathcal{I}$, computing a coefficient vector $\mathbf{a}$ such that $f_k \equiv \sum_{j=1}^{|\mathbf{s}_i|} a_j \cdot \mathbf{s}_i[j]$ is equivalent to checking if the polynomial $\mathbf{0}$ belongs to the span of polynomials $\mathbf{s}_i \cup \{f_k\}$.

Figure 2: Procedure for moving certain polynomials f_k with all InTrusted **variables to** Trusted **set**

4.2 Rules for Moving Polynomials into the Trusted Set

We now describe five rules for reducing the complexity of a PPE instance, whereby we mean reducing the number of elements represented by polynomials not in the set Trusted. The first two rules are closely derived from [45].The more complex third, fourth and fifth rules are novel to this work and require the AND/OR/NOT/PPE logic we introduced.

4.2.1 **Rule 1: Simple move a polynomial with all** InTrusted **variables to** Trusted **set.** In Figure 2, we adapt Rule 1 from [45] to output a PPE circuit. Given a PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and an index $k \in [m]$, Rule 1 can possibly be applied if $k \notin$ Trusted and the polynomial $f_k \in \mathbf{f}$ consists only of variables $u_i \in$ InTrusted (these conditions are necessary, but not sufficient).

Lemma 4.1 (Correctness of Rule 1). *Let $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ be a PPE problem instance as in Definition 3.1 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.*

The proof of the lemma follows from the correctness of the similar Rule 1 in [45].[6] We include this proof in Appendix B.

4.2.2 **Rule 2: Simple move of a polynomial with exactly one non-**InTrusted **variable to** Trusted **set.** In Figure 3, we recall Rule 2 from [45]; it does not need any changes for our purposes. Given a PPE problem $\Pi = (\mathbb{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and indices $j \in [n]$ and $k \in [m]$, Rule 2 can possibly be applied if $j \notin$ InTrusted, $k \notin$ Trusted and the polynomial $f_k \in \mathbf{f}$ is of the form $c \cdot u_j^d + h$, where the variable $u_j \in \mathbf{u}$, the polynomial h contains only variables in InTrusted, constant $c \in \mathbb{Z}_p^*$, and constant $d \in \mathbb{Z}_p$ s.t. d is relatively prime to $p - 1$.

Description of Rule 2 [45]

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

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

then proceed to the next step. Otherwise, abort and output $\perp$.
(2) Output $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$.

Figure 3: Procedure for moving certain polynomials f_k containing exactly one non-InTrusted **variable to** Trusted

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

The proof follows from the correctness of the similar Rule 2 in [45]. For completeness, we include it in Appendix C.

[6] Although [45] proved the correctness property for a weaker notion of PPE Testability, the proof can be easily adapted to our setting.

Description of SubstituteZero **Algorithm**

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

Output: A PPE Problem Π'.

- Construct vector $\mathbf{f}'$ of m polynomials as follows. For each $k \in [m]$, if $f_k = h_1 \cdot h + h_2$ for some polynomials h_1 and h_2, such that (1) h_2 does not have h as a factor, and (2) the number of monomials in h_2 is less than f_k when expressed in canonical form, then set $f'_k = h_2$. Otherwise, set $f'_k = f_k$.
- Output $\Pi' = (\mathcal{G}, n, m, \mathbf{f}', \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$.

Figure 4: Algorithm for updating a PPE problem instance when a specified polynomial h is set to 0.

4.2.3 **Rule 3: More general move of a polynomial with all** InTrusted **variables to** Trusted **set.** Rule 3 is a novel extension of Rule 1 (see Figure 2) for moving an untrusted polynomial f_k to the trusted set. Let us first observe some drawbacks in Rule 1 when the untrusted polynomial is in the group $\mathbb{G}_1$ or $\mathbb{G}_2$. For simplicity, let us consider execution of Rule 1 on input (Π, k) such that $\alpha_k = 1$. The Rule 1 algorithm explores a space of PPEs of the form $e(F_k, g_2) = \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j}$ where constants a_j are in $\mathbb{Z}_p$. In other words, Rule 1 only explores PPEs where the untrusted element F_k is paired with the generator of $\mathbb{G}_2$. Hohenberger and Vusirikala [45] observed that such a small class of PPEs appears insufficient to validate the proofs of the Dodis VRF [34] or the private keys of the Boyen-Waters IBE [30]. One could think of a natural extension of Rule 1 which explores a larger space of PPEs, where F_k is paired with some function of trusted polynomials, i.e., PPEs of the form $e(F_k, \prod_{j=1}^{|\mathbf{s}_2|} U_2[j]^{b_j}) = \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j}$, where constants a_j, b_j are in $\mathbb{Z}_p$. Such an extended algorithm computes constant vectors $\mathbf{a}, \mathbf{b}$ such that $f_k \cdot (\sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]) \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j]$ and outputs the circuit corresponding to PPE: $e(F_k, \prod_{j=1}^{|\mathbf{s}_2|} U_2[j]^{b_j}) = \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j}$.

However, this extension introduces a technical issue. Consider the PPE problem instance $\Pi = (\mathbb{G}, n = 2, m = 7, \mathbf{f} = \{1, 1, 1, x^3, y, xy, x\}, \alpha = \{1, 2, T, 1, 2, T, 1\}, \mathbf{u} = \{x, y\}, \text{Trusted} = \{1, 2, 3, 4, 5, 6\})$. Here, x, y are InTrusted variables and the only untrusted polynomial is x in group $\mathbb{G}_1$. On input $(\Pi, k = 7)$, the above extended rule outputs the PPE $e(F_7, F_5) = F_6$ and moves the only untrusted polynomial to the trusted set. Surprisingly, $e(F_7, F_5) = F_6$ is not a PPE testing circuit for the problem Π. Consider the following PPE challenge $(g_1, g_2, g_T, g_1^8, g_2^0, g_T^0, g_1^5)$. This is clearly a NO challenge, as there is NO x such that $x^3 = 8$ and $x = 5$ simultaneously. However, the challenge satisfies the PPE $e(F_7, F_5) = F_6$. Intuitively, the issue occurs because for the given PPE challenge $F_5 = g_2^0$ and therefore the PPE $e(F_7, F_5) = F_6$ does not validate the element F_7.

More generally, suppose the rule outputs $e(F_k, \prod_{j=1}^{|\mathbf{s}_2|} U_2[j]^{b_j}) = \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j}$. For the PPE challenges in which $\prod_{j=1}^{|\mathbf{s}_2|} U_2[j]^{b_j}$ evaluates to g_2^0, the PPE does not validate the correctness of F_k. To resolve the issue, our Rule 3 computes a PPE (from a larger class of PPEs than pairing only with a generator), and the resultant *testing circuit* is designed so that whenever the exponent of the paired element evaluates to zero, the logic of the testing circuit handles it properly. In Section 5, we show that this generalization of Rule 1 is very useful for the automated verification of the Dodis VRF proofs [34] and the Boyen-Waters IBE private keys [30].[7]

Rule 3 is formally described in Figure 5, its correctness property is captured in Lemma 4.3 and the proof of this lemma appears in Appendix D.

Lemma 4.3 (Correctness of Rule 3). *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ *be a PPE problem instance as in Definition 3.1 and let* $k \in [m]$. *Suppose* $\perp \neq (\text{IsIdentity}, C, \Pi', \Pi'') = \text{Rule3}(\Pi, k)$. *Then, for every pair of testing circuits* C' *and* C'' *for* Π' *and* Π'' *respectively, the PPE circuit* $Z := ((\text{NOT IsIdentity}) \wedge C \wedge C') \vee (\text{IsIdentity} \wedge C'')$ *is a testing circuit for* Π.

4.2.4 **Rule 4: General move of a polynomial with multiple non-**InTrusted **variables to the** Trusted **set.** In Figure 6, we describe a new Rule 4, which is an extension of Rule 2 for moving an untrusted polynomial f_k which has a variable in InTrusted[8] set to the Trusted set of polynomials. Recall that in order to apply Rule 2 to a polynomial f_k, the coefficient of the non-InTrusted variable u_j needs to be a non-zero constant. Hohenberger et al. [45] observed that this restriction to a constant coefficient appears insufficient to validate the private keys of the Boyen-Waters IBE [30]. One could naturally think of extending Rule 2 by allowing the coefficient of variable u_j to be an arbitrary polynomial h of InTrusted variables. However, if the polynomial h evaluates to 0 for a given set of InTrusted variables, then this becomes an issue because u_j is now zero-ed out in the f_k polynomial. We design our Rule 4 to validate this larger class of untrusted polynomials and the resulting *testing circuit* checks if the coefficient of the non-InTrusted variable u_j evaluates to 0 and handles it accordingly. Below, we further generalize this concept to validate untrusted polynomials with *multiple* non-InTrusted variables. Rule 4 is formally described in Figure 6 and prove its correctness property in Lemma 4.4.

Lemma 4.4 (Correctness of Rule 4). *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ *be a PPE problem instance as in Definition 3.1,* $j \in [n]$ *and* $k \in [m]$. *Suppose* $\perp \neq (\text{IsIdentity}, \Pi', \Pi'') = \text{Rule4}(\Pi, j, k)$. *Then for every pair of testing circuits* C' *and* C'' *of problem instances* Π' *and* Π'' *respectively, the PPE circuit* $((\text{NOT IsIdentity}) \wedge C') \vee (\text{IsIdentity} \wedge C'')$ *is a testing circuit for* Π. (Proof of this lemma appears in Appendix E.)

4.3 Applying the Rules

Rules 1-4 are combined into the main algorithm, called QSearch, in Figure 7 that takes as input a PPE problem and outputs a PPE circuit or the special message unknown. We prove that if QSearch outputs a testing circuit, then that circuit is guaranteed to correctly classify PPE challenges for this PPE problem.

[7] The issue with verification not working due to an exponent evaluating to zero is not a security issue in [30, 34], because this event happens with negligible probability when the public parameters are honestly generated. However, our definition of a testing circuit requires *perfect* correctness and therefore, we must check for and properly address this "zero" case.

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

Description of Rule 3

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ 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 Rule3(Π, k):

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

(2) Let the formal variables $F_1, F_2, \cdots, F_m$ represent group elements of any PPE challenge corresponding to Π. These formal variables also represent the input wires of the PPE circuits IsIdentity and C being constructed.

(3) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{f}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \le |\mathbf{s}_i|$, let $S_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$, and let $U_i[j]$ be the pairing product term computing $S_i[j]$ in terms of formal variables $F_1, \cdots, F_m$.

(4) Let $\overline{\alpha_k} = 3 - \alpha_k$. Check if there exist constant vectors $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{s}_T|})$ and $\mathbf{b} = (b_1, \cdots, b_{|\mathbf{s}_{\overline{\alpha_k}}|})$ with entries in $\mathbb{Z}_p$ s.t. $(\sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j])$ is not a constant polynomial when expressed in canonical form, and

$$f_k \cdot (\sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]) \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j].$$

(Computing coefficient vectors **a, b** reduces to checking if the polynomial **0** belongs to the span of polynomials $\mathbf{s}_T \cup f_k \cdot \mathbf{s}_{\overline{\alpha_k}}$.)

(5) If such (**a**, **b**) exist, then

- Compute PPEs

$$A := (\prod_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} U_{\overline{\alpha_k}}[j]^{b_j} = \mathbf{I}_{\overline{\alpha_k}}),$$

$$B := \prod_{j=1}^{|\mathbf{s}_T|} U_T[j]^{a_j} = \begin{cases} e(F_k, \prod_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} U_{\overline{\alpha_k}}[j]^{b_j}) & \text{if } \alpha_k = 1 \\ e(\prod_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} U_{\overline{\alpha_k}}[j]^{b_j}, F_k) & \text{if } \alpha_k = 2 \end{cases}$$

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

- Compute $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$ and $\Pi'' = \text{SubstituteZero}(\Pi, \sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j])$, where the SubstituteZero algorithm is described in Figure 4. (Intuitively, SubstituteZero creates a new PPE problem instance by substituting $\sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]$ 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 PPE problems Π', Π''.

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

Figure 5: Procedure for moving certain polynomials f_k with all InTrusted **variables to** Trusted

Theorem 4.5 (Correctness of the PPE Circuit Searching Algorithm in Figure 7). *Let* $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ *be a PPE problem instance as in Definition 3.1. Let* $C = \text{QSearch}(\Pi)$. *If* $C \neq$ unknown, *then* C *is a PPE testing circuit for* Π *as in Definition 3.5, and therefore* Π *is circuit testable.*

Proof of this theorem appears in Appendix F.

5 IMPLEMENTATION

We implemented the PPE circuit searching algorithm described in Figure 7 in a software tool called AutoCircuitPPE. We ran the tool on several 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 PPE circuit testable. Fortunately, our tool was able to produce outputs for the two main schemes left open by the previous AutoPPE tool [45] and for some new schemes not studied in that prior work. We now present the design of the AutoCircuitPPE tool followed by its test case results and performance numbers.

5.1 AutoCircuitPPE Implementation

We implemented AutoCircuitPPE using Ocaml version 4.02.3. We built the code on top of the AutoPPE[9] tool (Hohenberger and Vusirikala [45]; ACM CCS 2019), which in turn utilizes some of the parsing tools and data structures (to store polynomials) of the Generic Group Analyzer (GGA) tool[10] (Barthe et al. [18]; CRYPTO 2014). We also used the SageMath package[11] to solve systems of linear equations and implemented the remaining logic ourselves.

The input format of AutoCircuitPPE is the same as the AutoPPE 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] Besides, the tool optionally takes as input information that allows the tool to help the user encode some cryptosystem parameters as a PPE problem

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

[10] https://github.com/generic-group-analyzer/gga

[11] https://www.sagemath.org/

[12] Unlike AutoPPE, our tool does not take fixed/unfixed variables as input, as we did not find this information to be necessary or useful.

[13] While this program input is in a slightly different format than Definition 3.1, we stress that it is the same information.

Description of Rule 4

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$ and integers $j \in [n]$ and $k \in [m]$.
Output: Either $\perp$ (meaning could not apply the rule) or a PPE circuit IsIdentity and two PPE problems Π', Π''.
Steps of Rule4(Π, j, k):

(1) If polynomial $f_k \in \mathbf{f}$ is of the form $h_1 \cdot u_j^d + h_2$, where
- $j \notin$ InTrusted, $k \notin$ Trusted,
- the polynomial h_1 contains only variables in InTrusted, and
- the polynomial h_2 contains any variables other than u_j, and
- the constant $d \in \mathbb{Z}_p$ is relatively prime to $p-1$,

then proceed to the next step. Otherwise, abort and output $\perp$.

(2) Let the formal variables $F_1, F_2, \cdots, F_m$ represent group elements of any PPE challenge for Π. These formal variables will also represent the input wires of the PPE circuit IsIdentity being constructed.

(3) Compute completion lists $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_T\} = \mathsf{CL}(\mathbf{f}^{\text{Trusted}})$. For any $i \in \mathcal{I}$ and $j \leq |\mathbf{s}_i|$, let $S_i[j] = g_{\alpha_i}^{\mathbf{s}_i[j]}$, and let $U_i[j]$ be the pairing product term computing $S_i[j]$ in terms of formal variables $F_1, \cdots, F_m$.

(4) If there exists an $\alpha \in \mathcal{I}$ (recall $\mathcal{I} = \{1, 2, T\}$ in the Type III setting) and a constant vector $\mathbf{a} = (a_1, a_2, \cdots, a_{|\mathbf{s}_\alpha|})$ with elements in $\mathbb{Z}_p$ such that $h_1 = \sum_{j=1}^{|\mathbf{s}_\alpha|} a_j \cdot \mathbf{s}_\alpha[j]$, then set PPE

$$A := \prod_{j=1}^{|\mathbf{s}_\alpha|} U_\alpha[j]^{a_j} = \mathbf{I}_\alpha,$$

where $\mathbf{I}_\alpha$ is the identity element of the group $\mathbb{G}_\alpha$. Then define IsIdentity = MakeCircuit($\mathcal{G}$, m, $\boldsymbol{\alpha}$, A).

(5) If IsIdentity is defined, then set PPE problem $\Pi' = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted} \cup \{k\})$ and $\Pi'' = \text{SubstituteZero}(\Pi, h_1)$, where the function SubstituteZero is defined in Figure 4. Output IsIdentity and Π', Π''.

(6) If IsIdentity is not defined, output $\perp$.

Figure 6: Procedure for moving certain polynomials f_k containing exactly one non-InTrusted **variable to** Trusted

instance. In particular, all trusted and untrusted elements (represented by polynomials) are bilinear group elements in $\mathbb{G}_1, \mathbb{G}_2$ or $\mathbb{G}_T$ and Definition 3.1 does not allow including an element in $\mathbb{Z}_p$ in either set. However, since it is not uncommon for schemes to contain elements in the $\mathbb{Z}_p$ domain as part of their public or private parameters, we implemented a workaround for those schemes similar to AutoPPE.[14] The tool runs the PPE circuit searching algorithm in Figure 7 along with a few optimizations implemented in AutoPPE such as computing completion list before applying all the rules. It outputs either a PPE circuit or the special symbol unknown. The PPE circuit computed by the QSearch algorithm is generally very large;

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

Main Algorithm for PPE Testing Circuit Search

Input: A PPE problem $\Pi = (\mathcal{G}, n, m, \mathbf{f}, \mathbf{u}, \boldsymbol{\alpha}, \text{Trusted})$.
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') and
    (ii) if $C' \neq$ unknown, then output the PPE circuit $Q := C$ AND C'.
Rule 2. For $k = 1$ to m and $j = 1$ to n,
  (a) Call z = Rule2(Π, j, k).
  (b) If $z = \Pi' \neq \perp$, then
    (i) Call C' = QSearch(Π') and
    (ii) if $C' \neq$ unknown, then output the PPE circuit $Q := C'$.
Rule 3. For $k = 1$ to m,
  (a) Call z = Rule3(Π, k).
  (b) If z = (IsIdentity, C, Π', Π'') $\neq \perp$, then
    (i) Call C' = QSearch(Π') and
    (ii) if $C' \neq$ unknown, then call C'' = QSearch(Π'') and
    (iii) if $C' \neq$ unknown and $C'' \neq$ unknown, then output the PPE circuit $Q :=$ ((NOT IsIdentity) AND C AND C') OR (IsIdentity AND C'').
Rule 4. For $k = 1$ to m and $j = 1$ to n,
  (a) Call z = Rule4(Π, j, k).
  (b) If z = (IsIdentity, Π', Π'') $\neq \perp$, then
    (i) Call C' = QSearch(Π'),
    (ii) if $C' \neq$ unknown, then call C'' = QSearch(Π'') and
    (iii) if $C' \neq$ unknown and $C'' \neq$ unknown, then output the PPE circuit $Q :=$ ((NOT IsIdentity) AND C') OR (IsIdentity AND C'').
Final. Otherwise, output unknown.

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

therefore we optimize the circuit by a few techniques such as computing common sub-circuits only once.[15]

The source code for AutoCircuitPPE comprises roughly 4K lines of Ocaml code. The input file to the tool consists of the type of pairing, set of trusted and untrusted polynomials. For the schemes in Table 1, this information can be expressed within 3-6 lines of code. In our experience, most pairing-based schemes can be encoded into this input format within a few minutes. The ease of converting a given pairing-based scheme into the input format for AutoCircuitPPE makes the tool highly practical and useful. The code for AutoCircuitPPE is publicly available at https://github.com/JHUISI/auto-tools.

[15]Note that recursive calls to QSearch with the same arguments result in common sub-circuits. However, common sub-circuits could occur even in other scenarios. As a simple example, one recursive call to QSearch may result in a sub-circuit of the form (NOT PPE_1) OR (PPE_1 AND PPE_2), and another recursive call to QSearch with different inputs may result in a sub-circuit ((NOT PPE_1) AND PPE_3) OR (PPE_1 AND PPE_4). In this case, (NOT PPE_1) is a common sub-circuit that can be evaluated only once.

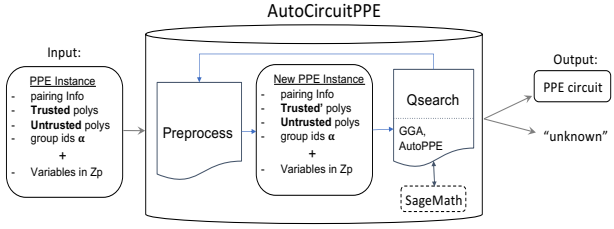

Figure 8: The workflow of the AutoCircuitPPE tool. The tool first preprocesses the problem instance, i.e., for every polynomial f in the trusted set and variable v in `Zp_vars`, the tool adds the polynomial $f \cdot v$ to the trusted set. It then runs the `QSearch` algorithm, which may output a new problem instance. AutoCircuitPPE preprocesses this instance before feeding it back to `Qsearch`. The tool utilizes and adapts portions of existing tools such as the GGA and AutoPPE for handling polynomials and completion sets and the SageMath package for solving systems of linear equations.

Input File Example

```
maps G1 * G1 ->GT.
Zp_vars [x1, x2].    (*input x*)
trusted_polys [F1 = a1*c, F2 = a2*c, F3 = c] in G1.    (*public key*)
untrusted_polys [F4 = (a1*x1 + (1-x1))] in G1.    (*proof*)
untrusted_polys [F5 = (a1*x1 + (1-x1)) * (a2*x2 + (1-x2))] in G1.
(*VRF output*)
```

Figure 9: Input file for Dodis VRF scheme when input length is 2.

5.2 A Detailed Example for the Dodis VRF (a [45] "gray area" scheme)

Let's walk through an example of using our tool on the Verifiable Random Function by Dodis [34]. The Setup algorithm takes as input security parameter λ and an input length n. It samples a Type I group $\mathcal{G} = (p, g_1, g_T, \mathbb{G}_1, \mathbb{G}_T, e)$, samples $c \leftarrow \mathbb{Z}_p$, $a_i \leftarrow \mathbb{Z}_p$ for $i \in [n]$. It then outputs secret key $\mathsf{sk} = (g_1, a_1, a_2, \cdots a_n)$ and verification key $\mathsf{vk} = (g_1, g_1^c, g_1^{c \cdot a_1}, g_1^{c \cdot a_2}, \cdots, g_1^{c \cdot a_n})$. The VRF algorithm takes as input secret key sk and bit string input x. It then outputs $y = g_1^{\prod_{i \text{ s.t. } x_i=1}^{n} a_i}$ and proof $\pi = \left(g_1^{\prod_{i \text{ s.t. } x_i=1}^{j} a_i}\right)_{j \in [n-1]}$. The goal of this example is to use this tool to generate a PPE circuit for verifying this VRF proof π; AutoPPE [45] output unknown on this scheme.

Figure 9 shows how to encode this scheme as input for AutoCircuitPPE. For space reasons, we will let $n = 2$. The pairing information is specified using the line `maps G1*G1->GT`, which denotes a Type I pairing[16]. The trusted set of polynomials (vk) along with their group identifiers are specified by `trusted_polys [_] in G_`, and the untrusted set of polynomials (VRF output) along with their group identifiers by `untrusted_polys[_] in G_`. For each polynomial, we also specify a formal variable `F_` which is used in the PPE circuit

[16] 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`.

Output of the Tool

Including polynomial F0 = 1 in trusted set of groups G1, GT.

Trusted set in G1: F1 = a1*c, F2 = a2*c, F3 = c
Untrusted set in G1: F4 = 1 - x1 + a1*x1, F5 = 1 - x1 - x2 + a1*x1 + a2*x2 + x1*x2 - a1*x1*x2 - a2*x1*x2 + a1*a2*x1*x2,
rule 3 on F4. isidentity := F3 = I C := e(F4,F3) = (e(F3,F0^x1))^-1*e(F1,F0^x1)*e(F0,F3)

Trusted set in G1: F1 = a1*c, F2 = a2*c, F3 = c, F4 = 1 - x1 + a1*x1
Untrusted set in G1: F5 = 1 - x1 - x2 + a1*x1 + a2*x2 + x1*x2 - a1*x1*x2 - a2*x1*x2 + a1*a2*x1*x2,
rule 3 on F5. isidentity := F3 = I C := e(F5,F3) = (e(F3^x2,F4))^-1*e(F2^x2,F4)*e(F3,F4)

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 1 - x1 + a1*x1
Untrusted set in G1: F5 = 1 - x1 - x2 + a1*x1 + a2*x2 + x1*x2 - a1*x1*x2 - a2*x1*x2 + a1*a2*x1*x2,
rule 4 on F5 and variable a2. isidentity := F4^x2 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 1 - x1 + a1*x1
Untrusted set in G1: F5 = 1 - x1 + a1*x1,
rule 1 on F5. C := F5 = F4

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0
Untrusted set in G1: F4 = 1 - x1 + a1*x1, F5 = 1 - x1 - x2 + a1*x1 + a2*x2 + x1*x2 - a1*x1*x2 - a2*x1*x2 + a1*a2*x1*x2,
rule 4 on F4 and variable a1. isidentity := F0^x1 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 1 - x1 + a1*x1
Untrusted set in G1: F5 = 1 - x1 - x2 + a1*x1 + a2*x2 + x1*x2 - a1*x1*x2 - a2*x1*x2 + a1*a2*x1*x2,
rule 4 on F5 and variable a2. isidentity := F4^x2 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 1 - x1 + a1*x1
Untrusted set in G1: F5 = 1 - x1 + a1*x1,
rule 1 on F5. C := F5 = F4

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0,
Untrusted set in G1: F4 = 1, F5 = 1 - x2 + a2*x2,
rule 1 on F4. C := F4 = F0

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 1
Untrusted set in G1: F5 = 1 - x2 + a2*x2,
rule 4 on F5 and variable a2. isidentity := F0^x2 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 1
Untrusted set in G1: F5 = 1,
rule 1 on F5 = 1. C := F5 = F0

Execution time : 1.682712s
((((NOT F3 = I) AND e(F4,F3) = (e(F3,F0^x1))^-1*e(F1,F0^x1)*e(F0,F3)) AND ((((NOT F3 = I) AND e(F5,F3) = (e(F3^x2,F4))^-1*e(F2^x2,F4)*e(F3,F4)) AND ACC) OR (F3 = I AND (((NOT F4^x2 = I) AND ACC) OR (F4^x2 = I AND (F5 = F4 AND ACC)))))) OR (F3 = I AND (((NOT F0^x1 = I) AND (((NOT F4^x2 = I) AND ACC) OR (F4^x2 = I AND (F5 = F4 AND ACC)))) OR (F0^x1 = I AND (F4 = F0 AND (((NOT F0^x2 = I) AND ACC) OR (F0^x2 = I AND (F5 = F0 AND ACC))))))))

List of gates after optimizing the circuit
G1 : F3 = I **G2** : e(F4,F3) = (e(F3,F0^x1))^-1*e(F1,F0^x1)*e(F0,F3)
G3 : e(F5,F3) = (e(F3^x2,F4))^-1*e(F2^x2,F4)*e(F3,F4)
G4 : F4^x2 = I **G5** : F5 = F4 **G6** : F0^x1 = I **G7** : F4 = F0
G8 : F0^x2 = I **G9** : F5 = F0 **G10** : NOT G1 **G11** : G10 AND G2 **G12** : G11 AND G3 **G13** : NOT G6 **G14** : NOT G4 **G15** : G4 AND G5 **G16** : G14 OR G15 **G17** : G13 AND G16 **G18** : NOT G8 **G19** : G8 AND G9 **G20** : G18 OR G19
G21 : G7 AND G20 **G22** : G6 AND G21 **G23** : G17 OR G22
G24 : G1 AND G23 **G25** : G12 OR G24

Figure 10: Output of the tool on Dodis VRF scheme when input length is 2.

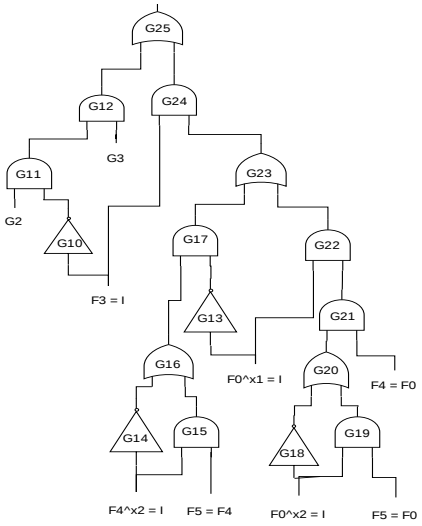

Figure 11: The PPE circuit output by AutoCircuitPPE **on the Dodis VRF for 2-bit Inputs. PPE gates G1-G9 (see Figure 10) are mentioned at input wires.**

output by the tool. Each bit of VRF input x is treated as a variable in $\mathbb{Z}_p$ and specified using `Zp_vars [_]`. Internally for every problem instance Π, for each trusted polynomial f and a $\mathbb{Z}_p$ variable x_i, the AutoCircuitPPE tool adds $x_i \cdot f$ to the trusted set[17]. Comments are specified with `(*....*)`.

Figure 10 shows AutoCircuitPPE's output the Figure 9 input. The tool applies the QSearch algorithm (see Section 4.3). For each recursive call made to QSearch, it prints the trusted and untrusted set polynomials, along with each rule applied. It then prints the PPE circuit output by QSearch. As this PPE circuit may contain some redundancy, the tool further optimizes the PPE circuit using simple tricks such as evaluating common sub-circuits only once and replacing a sub-circuit of the form x AND ((NOT x) OR y) with a circuit of the form (x AND y). The tool finally outputs the list of gates in the optimized PPE circuit, which we show pictorially in Figure 11.

5.3 Case Studies

We evaluated AutoCircuitPPE on various types of pairing-based schemes using a MacBook Pro 2015 laptop with 2.7GHz Intel Core i5 processor and 8GB 1867MHz DDR3 RAM. We present the results along with average execution times over 10 runs in Table 1. We retain AutoPPE's optimizations for computing PPEs in the rules efficiently. Lke AutoPPE, we simplified checking whether the constant d is relatively prime to $p-1$ in Rule 2 and 4, 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 a high degree on their variables.

Table 1 summarizes 29 test results. For IBE schemes, we ran our tool to output 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 that tests for the 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 problem instance similar to [45] (See [45] Section 5.2 for more details). As in [45], we encode the VRF bit string input of [34, 46, 47] schemes as a vector of $\mathbb{Z}_p$ variables. We observe that the size of the polynomials in these schemes grows exponentially in size with respect to the length of the 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 Type I, II and III pairing settings. Many of PPE problem instances in Table 1 are in the Type I setting (we first encoded in whatever setting the scheme's authors chose). We also translated several of these schemes into the Type III setting for testing. AutoCircuitPPE *outputs a PPE testing circuit for all the problem instances on which AutoPPE outputs a PPE testing set*[18]*. More importantly, our tool outputs PPE testing circuits for the Bellare-Kiltz-Peikert-Waters IBE [24], Dodis VRF [34], Boyen-Waters IBE [30] and some custom test cases on which AutoPPE was not able to produce a valid PPE testing set.*

We tested our tool on a few custom examples, some of them having more than 100 polynomials. The 100-DDH and 100-DBDH examples have already be tested in [45]. In the (new) DLIN test case, the trusted set contains polynomials $\{a, b, c, ax, by\}$ in group G_1, and the untrusted set contains the polynomial $c(x+y)$ in group $\mathbb{G}_1$. The 100-DBDH and DLIN examples are *not* PPE Testable under the Decisional Bilinear Diffie-Hellman (DBDH) assumption and DLIN assumptions respectively. We additionally designed two custom test cases meant to utilize all of our tool's rules. The results are in Table 1 and the details are in Appendix G.

Recall that AutoCircuitPPE optimizes the output of the QSearch algorithm (Section 4.3). Table 1 shows the number of PPE gates and Boolean gates output post-optimization. On the Bellare et. al. IBE scheme for 4-bit identities, Dodis VRF for 4-bit inputs and the Boyen-Waters IBE scheme, the PPE circuit output by QSearch has 98, 180 and 491 boolean gates, respectively, whereas post-optimization the corresponding PPE circuits have only 31, 49 and 124 boolean gates.

5.4 Open Problems

This work solves a major open problem posed by [45] by defining PPE circuits and developing a method for automatically generating them. We remark on two limitations of our tool, which are exciting directions for future research.

First, we do not handle rational polynomials; that is, our tool cannot accept inputs with elements of the form $g^{1/x}$,

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

[18] For all instances on which AutoPPE outputs a PPE testing set, our tool also outputs the same PPE circuit. This is because we retain Rules 1 and 2 used by AutoPPE and prioritize these rules over Rules 3 and 4 in our QSearch algorithm.

Scheme	Pairing	Type	AutoPPE output	PPE Circuit Testability	Our Tool Output	#PPE Gates	#Bool Gates	Run Time
Boneh-Franklin01 ([28])	Type I	IBE	Testable	Testable	Testable	1	0	1.63s
Gentry-Silverberg02 ([38])	Type I	IBE	Testable	Testable	Testable	1	0	1.54s
Boneh-Boyen04a ([26]) ($\ell = 160$)	Type I	HIBE	Testable	Testable	Testable	1	0	167s
Waters05 ([52]) ($\lvert H(\mathsf{id})\rvert = 16$)	Type I	IBE	Testable	Testable	Testable	1	0	10.92s
Naccache05 ([50]) ($\mathcal{B}(H(\mathsf{id})) = 8$)	Type III	IBE	Testable	Testable	Testable	1	0	1.62s
BBG05 ([27]) ($\ell = 8$)	Type I	HIBE	Testable	Testable	Testable	5	4	5.04s
Waters09 ([54])	Type I	IBE	Unknown	Not testable	Unknown	0	0	105.37s
BKPW12 ([24]) ($\lvert\mathsf{id}\rvert = 4$)	Type I	IBE	Unknown	Testable	Testable	14	31	4.57s
Boyen-Waters ([30])	Type I	Anon-IBE	Unknown	Testable	Testable	27	124	18.39s
BLS01 ([29])	Type I	Signature	Testable	Testable	Testable	1	0	1.69s
CL04 Scheme A ([32])	Type I	Signature	Testable	Testable	Testable	2	1	2.32s
CL04 Scheme B ([32])	Type I	Signature	Testable	Testable	Testable	4	3	1.60s
CL04 Scheme B ([32])	Type III	Signature	Testable	Testable	Testable	4	3	1.57s
CL04 Scheme C ([32]) ($\mathcal{B}(\mathsf{msg}) = 8$)	Type I	Signature	Testable	Testable	Testable	16	15	16.06s
Boyen-Waters ([31])	Type I	Signature	Testable	Testable	Testable	1	0	9.83s
AGOT14 ([3])	Type II	Signature	Testable	Testable	Testable	1	0	1.49s
Dodis ([34]) ($\lvert C(x)\rvert = 2$)	Type I	VRF	Unknown	Testable	Testable	9	16	1.68s
Dodis ([34]) ($\lvert C(x)\rvert = 4$)	Type I	VRF	Unknown	Testable	Testable	25	49	41.34s
Lys02 ([47]) ($\lvert C(x)\rvert = 5$)	Type I	VRF	Testable	Testable	Testable	5	4	109.81s
Lys02 ([47]) ($\lvert C(x)\rvert = 5$)	Type III	VRF	Testable	Testable	Testable	5	4	22.70s
Jager15 ([46]) ($\lvert H(x)\rvert = 4$)	Type I	VRF	Testable	Testable	Testable	5	4	26.05s
Jager15 ([46]) ($\lvert H(x)\rvert = 4$)	Type III	VRF	Testable	Testable	Testable	5	4	11.92s
RW13 ([51]) ($a = 60$)	Type I	CP-ABE	Testable	Testable	Testable	9	8	10.01s
RW13 ([51]) ($a = 8$)	Type III	CP-ABE	Testable	Testable	Testable	9	8	5.20s
100-DDH	Type I	Custom	Testable	Testable	Testable	1	0	4.99s
100-DBDH	Type I	Custom	Unknown	Not Testable	Unknown	0	0	4.48s
DLIN	Type I	Custom	Unknown	Not Testable	Unknown	0	0	1.2s
Custom Testcase 1 (Figure 12)	Type I	Custom	Unknown	Testable	Testable	9	16	4.76s
Custom Testcase 2 (Figure 14)	Type I	Custom	Unknown	Testable	Testable	6	9	3.58s

Table 1: The output of AutoCircuitPPE **on various PPE circuit testability problems. Here, ℓ represents the number of delegation levels in a HIBE scheme, $\lvert\mathsf{id}\rvert$ denotes the length of the identity, $\lvert H(\mathsf{id})\rvert$ 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.**

which rules out many interesting schemes such as the Gentry IBE [37], the Boneh-Boyen signatures [26] and the Dodis-Yampolskiy VRF [35]. A well-formedness test for such schemes should check if the denominators of the untrusted rational polynomials evaluate to 0, and output INVALID accordingly. While this was also an open problem from [45], we believe the general logic capabilities realized in this work create a foundation that could perform this check and then branch accordingly. Working this out, however, appears non-trivial.

Second, we'd like a more efficient method for encoding schemes for automated analysis. In the Dodis VRF scheme [34], the VRF function on input bit string x and private key $\{a_1, a_2, \cdots a_n\}$ outputs $g^{\prod^{n}_{i \text{ s.t. } x_i=1} a_i}$. In order to input the scheme to our tool, we encode the exponent polynomials as $\prod_{i=1}^{n}(a_i x_i + 1 - x_i)$. Notice that this polynomial has an exponential number of monomials incurring a huge computational cost for finding PPEs in our rules. This is notably the case in other VRF schemes [46, 47] as well. As a result, we could test the schemes only on small input lengths.

6 CONCLUSION

Computer automation holds great promise for improving the speed, accuracy and capabilities of cryptographic implementations. This work presents an automation algorithm and software tool for designing (pairing-based) cryptographic verification algorithms that can support arbitrary logic. The tool found verification algorithms for schemes that could not be handled by prior tools and for which we were unable to find solutions by hand. It executes quickly (usually 100 seconds or less) even for schemes with 100 or more elements in their description. There are many exciting future directions for automated cryptographic design and this tool will help in the automated design of algorithms requiring arbitrary logic.

ACKNOWLEDGMENTS

Susan Hohenberger was supported by NSF CNS-1414023, 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 grants CNS-1908611 and CNS-1414082, and the Packard Foundation. Brent Waters was supported in part by NSF CNS-1414082, CNS-1908611, Simons Investigator Award and Packard Foundation Fellowship.

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, 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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*.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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 2015*. 355–376.

[20] 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.

[21] 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.

[22] 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.

[23] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security*. 784–796.

[24] Mihir Bellare, Eike Kiltz, Chris Peikert, and Brent Waters. 2012. Identity-Based (Lossy) Trapdoor Functions and Applications. In *EUROCRYPT*.

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

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

[27] 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.

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

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

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

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

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

[33] 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*.

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

[35] 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)*.

[36] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. 2013. Attribute-Based Encryption for Circuits from Multilinear Maps. Cryptology ePrint Archive, Report 2013/128. http://eprint.iacr.org/.

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

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

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

[40] 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.

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

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

[43] 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*.

[44] 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.
[45] Susan Hohenberger and Satyanarayana Vusirikala. 2019. Are These Pairing Elements Correct? Automated Verification and Applications. In *ACM Conference on Computer and Communications Security*.
[46] Tibor Jager. 2015. Verifiable Random Functions from Weaker Assumptions. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC*. Springer, 121–143.
[47] Anna Lysyanskaya. 2002. Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In *Advances in Cryptology - CRYPTO*. Springer, 597–612.
[48] 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.
[49] Roberto Metere and Changyu Dong. 2017. Automated Cryptographic Analysis of the Pedersen Commitment Scheme. In *MMM-ACNS 2017*.
[50] David Naccache. 2005. Secure and *Practical* Identity-Based Encryption. *IACR Cryptology ePrint Archive* (2005). http://eprint.iacr.org/2005/369
[51] 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.
[52] Brent Waters. 2005. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*. Springer, 114–127.
[53] Brent Waters. 2009. Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In *CRYPTO*. Springer, 619–636.
[54] Brent Waters. 2009. Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In *CRYPTO*. Springer, 619–636.

A SHORTHAND NOTATIONS FOR CIRCUITS (CONT'D FROM SECTION 3.1)

In this section, we provide formal definitions for the shorthand notations introduced in Section 3.1.

- $\mathsf{MakeCircuit}(\mathcal{G}, m, \boldsymbol{\alpha}, P)$: 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 $N = 1, \text{Gates} = \{m+1\}$, $\text{out} = m+1, \text{GateType}(m+1) = (\text{PPE}, P), A = \emptyset, B = \emptyset$.
- C_{acc}: We use the notation C_{acc} to denote the circuit $\mathsf{MakeCircuit}(\mathcal{G}, m, \boldsymbol{\alpha}, P)$, where P is an always accepting PPE (for example, $g_1 = g_1$).

The formal definitions of the notations (C_1 AND C_2) and (C_1 OR C_2) are a little tricky as both C_1 and C_2 might share the same gate names. Henceforth, we first define the notation $\mathsf{Shift}(C, k)$ which renames the gates of a circuit C by an offset integer k. We then define (C_1 AND C_2), for example, by first shifting the gate names of C_2 by some offset k and then ANDing output wires of the circuits C_1 and $\mathsf{Shift}(C_2, k)$. Note that the input wires will remain the same.

- $\mathsf{Shift}(C, k)$: Given a circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}, \text{out}, \text{GateType}, A, B)$ and integer $k \geq 1$, the function $\mathsf{Shift}(C, k)$ 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 $\text{Gates}' = \{g+k : g \in \text{Gates}\}$, $\text{out}' = \text{out}+k$, $\text{GateType}'(g+k) = \text{GateType}(g)$, $A'(g+k) = A(g)$ and $B'(g+k) = B(g)$, whenever $A(g), B(g)$ are defined. Note: $\mathsf{Shift}(C, k)$ still has $\{1, 2, \cdots, m\}$ as the input wires.
- C_1 OP C_2 (where $\text{OP} \in \{\text{AND}, \text{OR}\}$): 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 $C_2' = \mathsf{Shift}(C_2, k) = (\mathcal{G}, m, \boldsymbol{\alpha}, N_2, \text{Gates}_2', \text{out}_2', \text{GateType}_2', A_2', B_2')$. 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 $\text{Gates}_1 \cup \text{Gates}_2'$, the set $\text{Gates} = \text{Gates}_1 \cup \text{Gates}_2' \cup \{\text{out}\}$, 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 a circuit $C = (\mathcal{G}, m, \boldsymbol{\alpha}, N, \text{Gates}, \text{out}, \text{GateType}, A, B)$, we use the notation NOT C to 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 $\text{Gates}' = \text{Gates} \cup \{\text{out}'\}$, 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.

B CORRECTNESS OF RULE 1

In this section, we prove the correctness of Rule 1 (Lemma 4.1). This proof is adapted from the proof of Rule 1 in [45].

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 a testing set for Π, then either:

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

We now analyze each of these cases.

In Case 1, we know that $C \wedge C'$ is not satisfied by the challenge $\mathbf{F}$. We take this in two subcases. First, suppose that $\mathbf{F}$ satisfies PPE C but not the circuit C'. This means that $\mathbf{F}$ is also a YES challenge for Π' (it can use the same settings for the variables), but for which C' is not satisfied. This contradicts the starting assumption that C' was a testing circuit for Π'. Second, suppose that $\mathbf{F}$ does not satisfy the PPE C. By definition of being a YES challenge, we know there exists an assignment to the variables $\mathbf{u}$ such that $F_i = g_{\alpha_i}^{f_i(\mathbf{u})}$ for all i. PPE C tests that F_k is equal to $g_{\alpha_i}^{f_i(\mathbf{u})}$, thus this equation being false contradicts the fact the $\mathbf{F}$ was a YES challenge.

In Case 2, since **F** is a NO challenge for Π where $C \wedge C'$ is satisfied, then **F** is also a NO challenge for Π' where C' is satisfied. We argue this as follows. By Definition 3.2 of a NO challenge for Π, there exists an assignment to $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}_p^n$ such that for all $i \in$ Trusted, $F_i = g_{\alpha_k}^{f_k(\mathbf{u})}$. To convert this to a NO challenge for Π', we also need to show that $F_k = g_{\alpha_k}^{f_k(\mathbf{u})}$ for this same assignment $\mathbf{u}$. This follows from the fact that PPE C is satisfied by this challenge and that C explicitly tests that F_k is computed this way, possibly with respect to an equivalent polynomial for $f_k \equiv \sum_{j=1}^{|\mathbf{s}_T|} a_j \cdot \mathbf{s}_T[j]$. Now since **F** 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 **F**, then C' is satisfied as well. This contradicts the original assumption that C' was a testing circuit for Π'. ∎

C CORRECTNESS OF RULE 2

We prove the correctness of Rule 2 (Lemma 4.3). This proof is adapted from the proof of Rule 2 in [45].

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. We first show that every YES challenge for Π is an YES challenge for Π', and similarly every NO challenge for Π is a NO challenge for Π'. This implies that every testing circuit C' for Π' is also a testing circuit for Π.

The PPE problems Π and Π' differ only in Trusted set, where the Π' set has the additional element $\{k\}$. As the definition of an YES challenge has no dependence on Trusted set, each YES challenge for Π is also an YES challenge for Π' and vice versa.

We now argue that any NO challenge for Π is also a NO challenge for Π'. Consider any NO challenge **F** for the PPE problem Π. By definition, **F** is not a YES challenge for Π (or, by the above, for Π'), and there exists an assignment of $\mathbf{u}^* \in \mathbb{Z}_p^n$ such that $F_i = g_{\alpha_i}^{f_i(\mathbf{u}^*)}$ $\forall i \in$ Trusted.

We want to show that $F_k = g_{\alpha_k}^{f_k(\mathbf{u}^*)}$. Since Rule2$(\Pi, j, k) \neq \perp$, we know that the polynomial f_k was of the form $c \cdot u_j^d + h$, according to the constraints of Rule2 i.e., the variable u_j is not used in any Trusted polynomials. Thus, for this setting of F_k in the challenge **F**, there exists only one setting of the variable $u_j \in \mathbb{Z}_p$ that is consistent with F_k being derived via the polynomial f_k and the settings of $u_i \in \mathbf{u}^*$. Let $F_k = g^y$ for some $y \in \mathbb{Z}_p$. Then let

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

There is a unique solution to the above since d is relatively prime to $p-1$. Thus, for setting of variables $\mathbf{u}^* \in \mathbb{Z}_p^n$ with u_j^* substituted with u_j, it holds that $F_i = g_{\alpha_i}^{f_i(\mathbf{u}^*)}$ $\forall i \in$ (Trusted $\cup$ $\{k\}$). Therefore **F** is a NO challenge for Π'. ∎

D CORRECTNESS OF RULE 3

In this section, we prove the correctness of Rule 3 (Lemma 4.3).

Proof. We observe that every challenge for Π is also a challenge for Π' and Π'', as they all share same group structure, the number of elements of m, and the group indicator vector α. Consider any testing circuits C', C'' for Π', Π'' respectively and any PPE challenge $\mathbf{F} = (F_1, F_2, \cdots F_m)$. We prove that if **F** is a YES challenge for Π, then it satisfies circuit Z defined above, and if **F** is a NO challenge for Π, it does not satisfy the circuit Z. We organize the proof into four cases.

Case 1 (**F** is an YES challenge for Π & doesn't satisfy IsIdentity):
In this case, by definition, there exists an assignment of variables **v** such that $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for all $\ell \in [m]$. We know that $\{g_{\alpha_\ell}^{f_\ell(\mathbf{v})})\}_\ell$ satisfies the circuit C for any variable assignment **v**. Therefore, the challenge **F** also satisfies the circuit C. We also observe that **F** is a YES challenge for Π'. This is because Π and Π' have the same set of polynomials $\{f_j\}_{j\in[m]}$ and only differ in the Trusted set. As a result, **F** satisfies the circuit (NOT IsIdentity) $\wedge\, C \wedge C'$, thus satisfying Z.

Case 2 (**F** is an YES challenge for Π & satisfies IsIdentity):
In this case, we want to show that **F** is a YES challenge for Π''. We know that $f_\ell = h_\ell \cdot (\sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]) + f_\ell''$ for some polynomial h_ℓ and other values as computed in Rule 3, where f_ℓ'' was replaced as the ℓ^{th} polynomial in Π'', due to $(\sum_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]) = 0$ via the fact that IsIdentity is satisfied. Consider any assignment of variables **v** s.t. $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$, $\forall \ell \in [m]$. We know that $\prod_j U_{\overline{\alpha_k}}[j]^{b_j} = I_{\overline{\alpha_k}}$ and therefore $\sum_j b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]$ evaluates to 0 for the variable assignment **v**. This implies, $F_\ell = g_{\alpha_\ell}^{f_\ell''(\mathbf{v})}$ for each $\ell \in [m]$. Therefore, **F** is an YES instance for Π'' and satisfies the circuit IsIdentity $\wedge\, C''$, thus satisfying Z.

Case 3 (**F** is a NO challenge for Π & doesn't satisfy IsIdentity):
Since we assume **F** does not satisfy the circuit IsIdentity in this case, we focus only on whether **F** satisfies $C \wedge C'$. By definition, **F** 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 **F** satisfies C, then **F** is a NO instance for Π'. Consider any assignment of variables **v** such that $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. If **F** satisfies C, it means $F_k = g_{\alpha_k}^{f_k(\mathbf{v})}$[19]. Consequently, $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for each $\ell \in$ Trusted $\cup$ $\{k\}$, and Π' is a NO instance. Therefore, **F** does not simultaneously satisfy the circuits $C \wedge C'$ and IsIdentity, and thereby does not satisfy Z.

Case 4 (**F** is a NO challenge for Π & satisfies IsIdentity):
In this case, we argue that **F** is a NO challenge for Π''. We know that for each $\ell \in [m]$, $f_\ell = h_\ell \cdot (\sum_j b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j]) + f_\ell''$, for

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

some polynomial h_ℓ, where f''_ℓ is the ℓ^{th} polynomial in Π''. Consider any assignment of variables $\mathbf{v}$ such that $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As $(\prod_{j=1}^{|\mathbf{s}_{\overline{\alpha_k}}|} U_{\overline{\alpha_k}}[j]^{b_j}) = I_{\overline{\alpha_k}}$, the polynomial $(\sum_j b_j \cdot \mathbf{s}_{\overline{\alpha_k}}[j])$ evaluates to 0 for the variable assignment $\mathbf{v}$. Therefore, $F_\ell = g_{\alpha_\ell}^{f''_\ell(\mathbf{v})}$ for each $\ell \in$ Trusted. Moreover, $\mathbf{F}$ cannot be an YES instance for Π''. This is because if there a variable assignment $\mathbf{v}$ such that $F_\ell = g_\ell^{f''_\ell(\mathbf{v})}$ for each $\ell \in [m]$, that would mean $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for each $\ell \in [m]$ which contradicts our initial assumption that $\mathbf{F}$ is a NO instance for Π. Therefore, $\mathbf{F}$ does not satisfy the circuits NOT IsIdentity and C'', and thereby does not satisfy Z.

■

E CORRECTNESS OF RULE 4

In this section, we prove the correctness of Rule 4 (Lemma 4.4).

Proof. We first observe that every valid challenge for Π is also a valid challenge for Π' and Π'', as they all share the same group structure, the number of elements m, and the group indicator vector $\boldsymbol{\alpha}$. Consider any testing circuits C', C'' for Π', Π'' respectively and any PPE challenge $\mathbf{F} = (F_1, F_2, \cdots F_m)$. We prove that if $\mathbf{F}$ is a YES challenge for Π, it satisfies circuit $Z := ((\text{NOT IsIdentity}) \wedge C') \vee (\text{IsIdentity} \wedge C'')$, and if $\mathbf{F}$ is a NO challenge for Π, it does not satisfy the circuit Z. We organize the proof into 4 cases.

Case 1 ($\mathbf{F}$ is a YES challenge for Π & doesn't satisfy IsIdentity):
We first observe that $\mathbf{F}$ is also a YES challenge for Π', as Π and Π' have the same set of polynomials $\{f_\ell\}_{\ell\in[m]}$ and only differ in the Trusted set. As a result, $\mathbf{F}$ satisfies the circuit (NOT IsIdentity) $\wedge\, C'$, and thus satisfies Z.

Case 2 ($\mathbf{F}$ is a YES challenge for Π & satisfies IsIdentity):
In this case, we argue that $\mathbf{F}$ is an YES challenge for Π''. Consider any assignment of variables $\mathbf{v}$ s.t. $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$, $\forall \ell \in [m]$. We know that $f_\ell = \beta_\ell \cdot h_1 + f''_\ell$ for some polynomial β_ℓ, where f''_ℓ is ℓ^{th} polynomial in Π''. As $\mathbf{F}$ satisfies the circuit IsIdentity, h_1 evaluates to 0 on input variable assignment $\mathbf{v}$. Consequently, $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})} = g_{\alpha_\ell}^{f''_\ell(\mathbf{v})}$ for each $\ell \in [m]$, and $\mathbf{F}$ is an YES instance for Π'' and satisfies the circuit IsIdentity $\wedge\, C''$, and thus satisfies Z.

Case 3 ($\mathbf{F}$ is a NO challenge for Π & doesn't satisfy IsIdentity):
In this case, $\mathbf{F}$ is not a YES instance for Π' as Π and Π' share the same set of polynomials. Also, there exists an assignment of InTrusted variables $\{v_i\}_{i\in\text{InTrusted}}$ such that $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As $\mathbf{F}$ does not satisfy the circuit IsIdentity, h_1 does not evaluate to 0 on variable assignment $\{v_i\}_{i\in\text{InTrusted}}$, and therefore for every possible value of F_k and h_2, there exists a value of u_j such that $F_k = g_{\alpha_k}^{h_1\cdot u_j + h_2}$. Consequently, there exists a variable assignment $\mathbf{v}$ such that $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$, $\forall \ell \in$ Trusted $\cup\, \{k\}$, and therefore $\mathbf{F}$ is a NO challenge for Π' and does not satisfy C'. Because it does not satisfy C' or IsIdentity, it cannot satisfy Z.

Case 4 ($\mathbf{F}$ is a NO challenge for Π & satisfies IsIdentity):
We know that for any $\ell \in [m]$, $f_\ell = \beta_\ell \cdot h_1 + f''_\ell$ for some polynomial β_ℓ, where f''_ℓ is the ℓ^{th} polynomial of Π''. Let $\mathbf{v}$ be any variable assignment such that $F_\ell = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for all $\ell \in$ Trusted. As $\mathbf{F}$ satisfies the circuit IsIdentity, h_1 evaluates to 0 on variable assignment $\mathbf{v}$, and $F_\ell = g_{\alpha_\ell}^{f''_\ell(\mathbf{v})}$, $\forall \ell \in$ Trusted. Furthermore, $\mathbf{F}$ cannot be an YES instance for Π''. This is because if $\mathbf{F}$ is an YES for Π'', there exists a variable assignment $\mathbf{v}$ such that $F_\ell = g_{\alpha_\ell}^{f''_\ell(\mathbf{v})} = g_{\alpha_\ell}^{f_\ell(\mathbf{v})}$ for all $\ell \in [m]$, which contradicts our assumption that $\mathbf{F}$ is a NO instance for Π. Therefore, $\mathbf{F}$ is a NO challenge for Π'' and does not satisfy (NOT IsIdentity) or C'', thus it cannot satisfy Z.

■

F QSEARCH: CORRECTNESS AND EFFICIENCY

We prove the correctness of QSearch algorithm (Theorem 4.5).

Proof. We provide a sketch of how to prove this theorem by induction on the number of untrusted polynomials and the total number of monomials in all the polynomials of $\mathbf{f}$. The critical correctness arguments required have already been covered for each rule in Lemmas 4.1 to 4.4. When QSearch is invoked on Π with either zero untrusted polynomials or 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 $\mathbf{f}$. It must have invoked one of the 4 rules. By Lemmas 4.1 to 4.4 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 the PPE problem Π. ■

Efficiency of QSearch. We now turn to the asymptotic complexity of the QSearch algorithm. A call to QSearch scans all the untrusted polynomials to check if any rule is applicable, and then calls QSearch recursively at most 2 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

Input to the Tool

maps G1 * G1 ->GT.
trusted_polys [F1 = x, F2 = x*a, F3 = x*b, F4 = x*c] in G1.
untrusted_polys [F5 = r, F6 = x*r, F7 = a, F8 = a+b*r, F9 = a*c+d*r] in G1.

Figure 12: Input to the tool on our custom testcase 1.

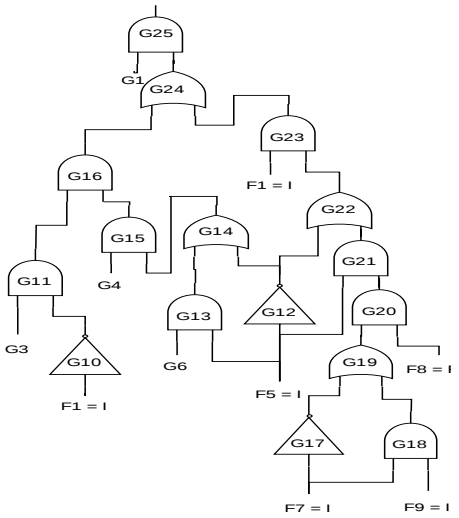

Figure 13: The PPE circuit output by our AutoCircuitPPE **tool on Figure 12 testcase**

multiplications taking $O(m^2 \cdot s^2)$ time. Checking if $\mathbf{0}$ lies in the span of $O(m^2)$ polynomials (number of polynomials in the completion lists) each having at most $O(s^2)$ monomials involves solving a system of $O(m^2 \cdot s^2)$ linear equations (upper bound on the number of monomials in the completion list) each of size $O(m^2)$, which takes at most $O((m^2 \cdot s^2)^\omega)$ time, where n^ω is the complexity of multiplying two $n \times n$ matrices. Executing Rules 2 and 4 additionally involves checking if an untrusted polynomial is in the desired format which takes at most $O(s)$ time. Therefore, applying all the rules to all the untrusted polynomials takes at most $O(m \cdot (ms)^{2\omega})$ time.

Now let us compute the total number of times the QSearch algorithm is called recursively. Suppose QSearch is run on problem Π and suppose it triggers a rule that outputs two PPE problems Π_1 and Π_2. (Rules 1 and 2 output only one PPE problem, but we can treat them as Rules 3 and 4 respectively in the worst case.) The problem Π_1 is obtained by moving an untrusted polynomial to the trusted set, and the problem Π_2 is obtained by substituting some polynomial by zero. Note that Π_2 cannot be equal to the original problem as Rule3 outputs $\perp$ otherwise, and Rule 4 substitutes some monomial for zero. Therefore some polynomial of Π_2 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 $O(m \cdot (ms)^{2\omega} \cdot 2^{m+k})$ time.

Input to the Tool

maps G1 * G1 ->GT.
trusted_polys [F1 = x, F2 = x*a, F3 = x*b] in G1.
untrusted_polys [F4 = r, F5 = x*r, F6 = a, F7 = a+b*r] in G1.

Figure 14: Input to the tool for our custom testcase 2.

Even though our algorithm has high theoretical complexity, in Section 5 we show that it runs reasonably fast for many real-world schemes. This is due to the fact that when we call the SubstituteZero function to substitute a polynomial with zero, the resulting problem usually has a much smaller size.

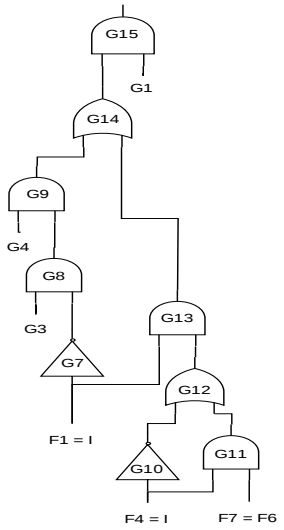

Figure 15: The PPE circuit output by our AutoCircuitPPE **tool on custom testcase 2 shown in Figure 14**

G CUSTOM TESTCASES

We demonstrate AutoCircuitPPE on some custom test cases.

Custom Testcase 1. The input to the tool is described in Figure 12. The output of the tool is displayed in Figure 17. For space reasons, we show only the optimized PPE circuit computed by the tool. It is interesting to note that AutoCircuitPPE utilizes all our 4 rules to construct a PPE circuit. We plot the PPE circuit in Figure 13.

Custom Testcase 2. The input to the tool is described in Figure 14. The output of the tool is displayed in Figure 16. We plot the PPE circuit in Figure 15.

Output of the Tool

Trusted set in G1: F0 = 1, F1 = x, F2 = a*x, F3 = b*x,
Untrusted set in G1: F4 = r, F5 = r*x, F6 = a, F7 = a + b*r,
rule 2 applied to F4 = r.

Trusted set in G1: F0 = 1, F1 = x, F2 = a*x, F3 = b*x, F4 = r,
Untrusted set in G1: F5 = r*x, F6 = a, F7 = a + b*r,
rule 1 applied to F5 = r*x. C := e(F5,F0) = e(F1,F4)

Trusted set in G1: F0 = 1, F1 = x, F2 = a*x, F3 = b*x, F4 = r, F5 = r*x,
Untrusted set in G1: F6 = a, F7 = a + b*r,
rule 3 applied on F6 = a. isidentity := F1 = I C := e(F6,F1) = e(F0,F2)

Trusted set in G1: F0 = 1, F1 = x, F2 = a*x, F3 = b*x, F4 = r, F5 = r*x,
F6 = a,
Untrusted set in G1: F7 = a + b*r,
rule 3 applied on F7 = a + b*r. isidentity := F1 = I C := e(F7,F1) =
e(F3,F4)*e(F1,F6)

Trusted set in G1: F0 = 1, F1 = 0, F2 = 0, F3 = 0, F4 = r, F5 = 0, F6 = a,
Untrusted set in G1: F7 = a + b*r,
rule 4 applied on F7 = a + b*r and variable b. isidentity := F4 = I

Trusted set in G1: F0 = 1, F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = a,
Untrusted set in G1: F7 = a,
rule 1 applied to F7 = a. C := F7 = F6

Trusted set in G1: F0 = 1, F1 = 0, F2 = 0, F3 = 0, F4 = r, F5 = 0,
Untrusted set in G1: F6 = a, F7 = a + b*r,
rule 2 applied to F6 = a.

Trusted set in G1: F0 = 1, F1 = 0, F2 = 0, F3 = 0, F4 = r, F5 = 0, F6 = a,
Untrusted set in G1: F7 = a + b*r,
rule 4 applied on F7 = a + b*r and variable b. isidentity := F4 = I

Trusted set in G1: F0 = 1, F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = a,
Untrusted set in G1: F7 = a,
rule 1 applied to F7 = a. C := F7 = F6

Execution time : 1.857338s

List of gates after optimizing the circuit
G1 : e(F5,F0) = e(F1,F4) **G2** : F1 = I **G3** : e(F6,F1) = e(F0,F2)
G4 : e(F7,F1) = e(F3,F4)*e(F1,F6) **G5** : F4 = I **G6** : F7 = F6
G7 : NOT G2 **G8** : G7 AND G3 **G9** : G8 AND G4 **G10**
: NOT G5 **G11** : G5 AND G6 **G12** : G10 OR G11 **G13** :
G2 AND G12 **G14** : G9 OR G13 **G15** : G1 AND G14

Figure 16: Output of the tool on the custom testcase 2 shown in Figure 14.

Assigning FID 0 to every unit polynomial 1
Trusted set in G1: F1 = x, F2 = a*x, F3 = b*x, F4 = c*x,
Untrusted set in G1: F5 = r, F6 = r*x, F7 = a, F8 = a + b*r, F9 = a*c + d*r,
rule 2 applied to F5 = r.

Trusted set in G1: F1 = x, F2 = a*x, F3 = b*x, F4 = c*x, F5 = r,
Untrusted set in G1: F6 = r*x, F7 = a, F8 = a + b*r, F9 = a*c + d*r,
rule 1 applied to F6 = r*x. C := e(F6,F0) = e(F1,F5)

Trusted set in G1: F1 = x, F2 = a*x, F3 = b*x, F4 = c*x, F5 = r, F6 = r*x,
Untrusted set in G1: F7 = a, F8 = a + b*r, F9 = a*c + d*r,
rule 3 applied on F7 = a. isidentity := F1 = I C := e(F7,F1) = e(F0,F2)

Trusted set in G1: F1 = x, F2 = a*x, F3 = b*x, F4 = c*x, F5 = r, F6 = r*x, F7 = a,
Untrusted set in G1: F8 = a + b*r, F9 = a*c + d*r,
rule 3 applied on F8 = a + b*r. isidentity := F1 = I C := e(F8,F1) = e(F3,F5)*e(F1,F7)

Trusted set in G1: F1 = x, F2 = a*x, F3 = b*x, F4 = c*x, F5 = r, F6 = r*x, F7 = a, F8 = a + b*r,
Untrusted set in G1: F9 = a*c + d*r,
rule 4 applied on F9 = a*c + d*r and variable d. isidentity := F5 = I

Trusted set in G1: F1 = x, F2 = a*x, F3 = b*x, F4 = c*x, F5 = 0, F6 = 0, F7 = a, F8 = a,
Untrusted set in G1: F9 = a*c,
rule 3 applied on F9 = a*c. isidentity := F1 = I C := e(F9,F1) = e(F4,F8)

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = a, F8 = a,
Untrusted set in G1: F9 = a*c,
rule 4 applied on F9 = a*c and variable c. isidentity := F7 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = 0, F8 = 0,
Untrusted set in G1: F9 = 0,
rule 1 applied to F9 = 0. C := F9 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = a,
Untrusted set in G1: F8 = a + b*r, F9 = a*c + d*r,
rule 4 applied on F8 = a + b*r and variable b. isidentity := F5 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = a, F8 = a + b*r,
Untrusted set in G1: F9 = a*c + d*r,
rule 4 applied on F9 = a*c + d*r and variable c. isidentity := F7 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = 0, F8 = b*r,
Untrusted set in G1: F9 = d*r,
rule 4 applied on F9 = d*r and variable d. isidentity := F5 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = 0, F8 = 0,
Untrusted set in G1: F9 = 0, rule 1 applied to F9 = 0. C := F9 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = a,
Untrusted set in G1: F8 = a, F9 = a*c,
rule 1 applied to F8 = a. C := F8 = F7

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = a, F8 = a,
Untrusted set in G1: F9 = a*c,
rule 4 applied on F9 = a*c and variable c. isidentity := F7 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = 0, F8 = 0,
Untrusted set in G1: F9 = 0, rule 1 applied to F9 = 0. C := F9 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = a,
Untrusted set in G1: F8 = a + b*r, F9 = a*c + d*r,
rule 2 applied to F7 = a.

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = a,
Untrusted set in G1: F8 = a + b*r, F9 = a*c + d*r,
rule 4 applied on F8 = a + b*r and variable b. isidentity := F5 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = a, F8 = a + b*r,
Untrusted set in G1: F9 = a*c + d*r,
rule 4 applied on F9 = a*c + d*r and variable c. isidentity := F7 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = r, F6 = 0, F7 = 0, F8 = b*r,
Untrusted set in G1: F9 = d*r,
rule 4 applied on F9 = d*r and variable d. isidentity := F5 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = 0, F8 = 0,
Untrusted set in G1: F9 = 0, rule 1 applied to F9 = 0. C := F9 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = a, Untrusted set in G1: F8 = a, F9 = a*c,
rule 1 applied to F8 = a. C := F8 = F7

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = a, F8 = a, Untrusted set in G1: F9 = a*c,
rule 4 applied on F9 = a*c and variable c. isidentity := F7 = I

Trusted set in G1: F1 = 0, F2 = 0, F3 = 0, F4 = 0, F5 = 0, F6 = 0, F7 = 0, F8 = 0,
Untrusted set in G1: F9 = 0, rule 1 applied to F9 = 0. C := F9 = I

Execution time : 1.882642s
List of gates after optimizing the circuit
G1 : e(F6,F0) = e(F1,F5) **G2** : F1 = I **G3** : e(F7,F1) = e(F0,F2) **G4** : e(F8,F1) = e(F3,F5)*e(F1,F7) **G5** : F5 = I **G6** : e(F9,F1) = e(F4,F8) **G7** : F7 = I **G8** : F9 = I **G9** : F8 = F7 **G10** : NOT G2 **G11** : G10 AND G3 **G12** : NOT G5 **G13** : G5 AND G6 **G14** : G12 OR G13 **G15** : G4 AND G14 **G16** : G11 AND G15 **G17** : NOT G7 **G18** : G7 AND G8 **G19** : G17 OR G18 **G20** : G9 AND G19 **G21** : G5 AND G20 **G22** : G12 OR G21 **G23** : G2 AND G22 **G24** : G16 OR G23 **G25** : G1 AND G24

Figure 17: Output of the tool on the test case in Figure 12