Development and Implementation of Novel Pair Value Encryption System

Rushil Mallarapu*

June 6, 2021

Abstract

Multiparty cryptography (MPC) is a cryptographic paradigm that seeks to have multiple, mutually distrustful parties compute a joint function of their inputs. Homomorphic encryption (HE) is a related, but distinct paradigm that seeks to allow arbitrary computations to be lifted over encrypted data. In both cases, general versions of such protocols are highly complex, requiring multiple rounds of communication, introducing implementation weaknesses and overheads which limit their practical efficacy. This research merges the insights behind these paradigms to develop what we term Pair Value Encryption (PVE), a lightweight protocol for performing arbitrary Boolean comparisons over encrypted data originating from potentially distrustful parties. We propose an abstract definition for PVE systems, demonstrate a protocol implementing this specification, and prove this protocol is both correct and secure. Python implementation and an example use case in patient matching in healthcare systems demonstrate both the low overhead and real-world applicability of this paradigm.

Keywords — Multiparty Communication, Homomorphic Encryption, RSA

1 Introduction

(Need for communication) (MPC) (Homomorphic encryption) (Failings) (Introduce PVE) (Contributions)

Cryptographic systems are designed to facilitate sensitive communication between secret agents. In the majority of standard practical protocols, such as RSA or Diffie-Hellman, these secret agents are mutually trustful, but are seeking to protect their communications from malicious external observers.

Intelligence operations often require secret agents to communicate with other agents from different organizations. When two spies meet to exchange secrets, they use a type of secret handshake to ensure that the parties participating in the exchange are the ones intended. For example, an FBI agent may want to communicate only with CIA agents, and if this is not the case, the communication should drop without revealing membership information and why the communication failed. This form of live drop communication,1 when parties are online and interact, has been implemented in cryptography and it is referred to as secret handshake (SH) protocol [10]. In SH, two parties agree on the same secret key only if they are both from the same group. Privacy is preserved in the sense that, if the handshake fails, nobody learns anything relevant other than the participants are not in the same group. In SH with dynamic matching [7], groups and roles can even be determined just before the protocol execution.

Malicious software, or malware, is used for a variety of unauthorized actions, from information theft to targeted denial of computational resources. Malware is estimated to cost the national economy between $57 billion and $109 billion, making it imperative to develop novel technologies to mitigate its ability to achieve its goals [1]. For it to do so, malware needs to communicate with a command and control (CC) center, to relay instructions, collect information, and receive updates. Botmasters – CC controllers – used to hardcode IPs to allow communicating with their malware, leading to the connections being shut down upon discovery of the malware. To address this, malware engineers designed domain generation algorithms (DGAs) to allow CC communication with malware could persist even after detection.

Embedded into many forms of malware, domain generation algorithms work by randomly producing thousands of domain names and attempting to resolve all of them against its DNS server [2]-[3]. The botmaster only needs to have registered a few of these domain names. When the malware-embedded DGA generates a domain that the botmaster has registered, it is able to establish a valid IP address and communicate with the CC center. As such, detecting DGA

*rushil.mallarapu@gmail.com

domains presents an asymmetric challenge to defenders: with standard blacklisting, defenders would need to keep registering generatable domains, while attackers would only need one successful connection to communicate or execute malware [4]. DGA- based malware includes Conficker, Stuxnet (the malware such, cybersecurity methodologies require an intelligent way of detecting and responding to the use of DGAs by malware in order to assure network security.

The binary classification problem this research addresses – detect whether a domain is maliciously generated or a genuine address – is well-researched in information security [5]. Knowledge of whether a domain name itself is genuine or malicious is especially useful, as gathering contextual information can present an additional overhead or may be unavailable within certain situations. Some algorithms use WHOIS and NXDomain records to gather rough predictions on whether a domain name is DGA. More recently, Curtin et al. proposed an RNN architecture for detecting DGA domains based on domain name and WHOIS data, but they acknowledged that their model was limited by disputes regarding the ability for open access to domain metadata [6].

Standard techniques for DGA detection with machine learning over domain names involve classification based on engineered text features, but this makes the model both vulnerable to adversarial training and impeded in detecting classifiable boundaries when observed DGA features overlap with those of genuine domains [5]. The use of deep learning methodologies enables DGA detection algorithms to avoid the dependence on feature engineering, as such models learn implicit features. As such, active research on DGA detection focuses on developing various deep learning architectures suited to the task, see e.g. [5]-[7]. We report the development of DGAIntel, which implements a convolutional-recurrent architecture for binary DGA classification in a lightweight package easily deployable for usage by cybersecurity architects.

2 Theory

In this section, we describe the general PVE protocol, describe an algorithm achieving this protocol, prove that it provides the correct results, and finally, prove it it is as hard to break as RSA.

2.1 PVE Protocol

Pair value encryption derives from a biomimetic influence – the core idea behind the protocol reflects the behavior of protein folding. In biological systems, chains of amino acids fold and combine into proteins, which can recognize other substrates and proteins by way of steric (i.e. in real space) matching. However, the key amino acid chain is obfuscated by the complex, often hard-to-predict process of protein folding[1]. Thus, the PVE protocol was derived to function on a similar basis – the encryption step should obfuscate plaintext values, but the comparison step should combine two matched ciphertext values to reduce to a simple Boolean variable.

Consider an arbitrary set of potentially distrustful agents, each of has a secret value that they want to compare in some Boolean fashion to those of other agents. Let X be the *problem space*, or the set of possible values that an agent might take. Moreover, let $\phi : (X, X) \mapsto \{0, 1\}$ be a binary comparison function over X. Let Z be the *plaintext space*, or set of values used to encode the problem set[2]. This also implies the existance of an invertable mapping $\gamma : X \mapsto Z$, converting values in the problem space to the plaintext space.

The pair value encryption protocol is a set of three functions:

1. $e : (Z, K) \mapsto Z$ – encrypt values, based on an encryption key
2. $c : (Z, Z) \mapsto \{0, 1\}$ – compare encrypted values
3. $e : (Z, K) \mapsto Z$ – decrypt values, based on a decryption key

In order to preserve the structure of the scenario as between X and ϕ, which depend on the use case, and Z, which depends on the implementation, we require that γ be constructed such that the following lemma hold true.

Lemma 2.1 (Preservation)**.** *For all $x, y \in X$, let $x' \coloneqq \gamma(x)$ and $y' \coloneqq \gamma(y)$. Then $c(e(x'), e(y')) \iff \phi(x, y)$.*

2.2 Algorithm

The implementation we present of the PVE system is based on the mathematics of RSA. First, we describe the initialization procedure for the algorithm, which selects the encryption and decryption keys.

[1] Of interest in this analogy is the fact that the protein folding problem is NP-complete

[2] Encrypted values are also in Z – this has the benefit of meaning neither a malicious agent nor an outside observer can distinguish unencrypted from encrypted data

Algorithm 1: PVE Setup

Input: Two independent primes (p,q)
Result: PVE modulus (n), encryption key (e), decryption key (d)

$n \leftarrow p \cdot q$;
$\phi(n) \leftarrow (p-1) \cdot (q-1)$;
for $i \leftarrow Z$ **do**
  if $\gcd(i, \phi(n)) = 1$ **then**
    $e \leftarrow i$;
  end
end
$d \leftarrow e^{-1} \pmod{\phi(n)}$;

Output: (n,e,d)

Note that a necessary postcondition of this PVE initialization step is that $de \equiv 1 \pmod{\phi(n)}$. Now, we provide algorithms for the encryption, comparison, and decryption procedure.

Algorithm 2: PVE Encryption

Input: Problem text ($x \in X$)
Data: PVE modulus (n), encryption key (e)
Result: Ciphertext ($z' \in Z$)

$z \leftarrow \gamma(x)$;
$z' \leftarrow z^e \pmod{n}$;

Output: z'

Algorithm 3: PVE Comparison

Input: Two ciphertexts ($z, z' \in Z$)
Data: PVE modulus (n)
Result: Comparison ($cmp \in \{0, 1\}$)

$cmp \leftarrow z \cdot z' \pmod{n}$;

Output: cmp

Algorithm 4: PVE Decryption

Input: Ciphertext ($z' \in Z$), decryption key (d)
Data: PVE modulus (n)
Result: Problem text ($x \in X$)

$z \leftarrow z'^d \pmod{n}$;
$x \leftarrow \gamma^{-1}(z)$;

Output: x

Hello
Hello

Theorem 2.2. *Correctness of PVE Theory – Algorithm, Proof of Correctness, Proof of Reducability to Strong RSA*

3 Implementation

Discussion – Reflect on performance, ease of implementation, and pitfalls in both RSA and PVE
Implementation – Python code, examples

4 Conclusion

Use Euler's theorem to compute:

1. $5^{60} \pmod{21}$
2. $2^{35} \pmod{35}$

4.1

First, we verify that 5 and 21 are coprime, which they are. Then, we compute $\phi(21)$ *via the standard route of decomposing 21 as its prime factorization* $21 = 3 \cdot 7$. *Thus,* $\phi(21) = \phi(3)\phi(7) = 2 \cdot 6 = 12$. *Now, by Euler's theorem, it holds that* $5^{12} \equiv 1 \pmod{21}$. *Dividing 60 by 12 gives* $60 = 5 \cdot 12 + 0$, *by the division algorithm. Therefore,* $5^{60} \equiv (5^{12})^5 \equiv 1^5 \equiv 1 \pmod{21}$. *Thus,* $5^{60} \equiv 1 \pmod{21}$.

4.2

As before, we verify that 2 and 35 are coprime, which they are. Next, we compute $\phi(35)$ *by decomposing 35 as its prime factorization* $35 = 5 \cdot 7$. *Thus,* $\phi(35) = \phi(5)\phi(7) = 4 \cdot 6 = 24$. *Now, by Euler's theorem, it holds that* $2^{24} \equiv 1 \pmod{35}$. *Dividing 35 by 24 gives* $35 = 1 \cdot 24 + 11$, *by the division algorithm. Therefore,* $2^{35} \equiv (2^{24})^1 \cdot 2^{11} \equiv 1^1 \cdot 2^{11} \equiv 2048 \pmod{35}$. *From here, we can use the division algorithm to divide 2048 by 35, giving* $2048 = 58 \cdot 35 + 18$. *Thus,* $2^{35} \equiv 18 \pmod{35}$.

5

Find the last two digits of 123^{403}.

5.1

We want to find the value of $123^{403} \pmod{100}$. *First, we verify that 123 and 100 are coprime, as indeed they are. Now, we find* $\phi(100) = \phi(2^2 \cdot 5^2) = \phi(2^2)\phi(5^2) = (4-2)(25-5) = 2 \cdot 20 = 40$ *as per the usual method. Thus, by Euler's theorem, we have that* $123^{40} \equiv 1 \pmod{100}$. *Using the division algorithm, we have that* $403 = 10 \cdot 40 + 3$. *As such,* $123^{403} \equiv (123^{40})^1 0 \cdot 123^3 \equiv 1^1 0 \cdot 123^3 \equiv 123^3 \equiv 1860867 \pmod{100}$. *Finally, taking the last two digits of this, we have that* $123^3 \equiv 67 \pmod{100}$. *Thus, the final two digits of* 123^{403} *are 67.*

6

Alice chooses primes $p = 17$ *and* $q = 23$, *as well as public key 7. What is the RSA decryption exponent?*

6.1

Here, we start by applying the RSA cryptosystem to generate the modulus $n = pq$, *which here is* $n = 17 \cdot 23 = 391$. *We also compute* $\phi(n) = (p-1)(q-1) = 16 \cdot 22 = 352$ *Next, the public encryption key is* $e = 7$, *and it is trivial to verify that 7 and 352 are coprime. Now, we must find the decryption key d by solving* $ed \equiv 1 \pmod{\phi(n)}$, *or* $7d \equiv 1 \pmod{352}$.

To solve this, we start by knowing that $\gcd(7, 352)$ *is of course one, so there will be only one unique solution d. Now, we apply the extended Euclidean algorithm to (352, 7).*

$$\begin{aligned} &\gcd(352, 7) && 352 = 50 \cdot 7 + 2 \\ =&\gcd(7, 2) && 7 = 3 \cdot 2 + 1 \\ =&\gcd(2, 1) && 2 = 2 \cdot 1 + 0 \\ =&1 \end{aligned}$$

	x	y
352	1	0
7	0	1
2	1	-50
1	-3	151

Therefore, the solution to this congruence is $d \equiv 151 \cdot 1 \equiv 151 \pmod{352}$, *and it is easy to verify that* $7 \cdot 151 \equiv 1057 \equiv 1 \pmod{352}$. *Thus, the decryption exponent is* $d = 151$.