CYBERNETICA

Public

Actively Secure 2-Party Computation Protocols and Frameworks

Technical Report

Version 1.0

2022

ID D-2-501

1 Introduction

Many privacy sensitive data workflows consist of two parties separated by organizational or physical borders sharing data for joint computation. A secure two-party computation (2PC) framework is a desirable tool for swapping conventional data sharing for secure computation. In doing so, the parties can improve user privacy and comply with data protection regulations.

Secure two-party computation naturally fits into a client-server computation as a swap-in replacement. For instance, a 2PC framework could be used for mobile analytics with the mobile device acting as one of the two parties.

A secure data workflow should not leak secret data even in the presence of compromised hardware or software. An attacker with full access to the computer executing the data analysis should not be able to recover any secrets from the other parties. To ensure such a protection, a 2PC framework must have input privacy in the presence of an active adversary.

Security against active adversaries In this security setting, the corrupted party is allowed to deviate from the protocol to attempt to learn secrets and affect the output. When the corrupted party misbehaves, the honest party should be able to halt execution and the honest party's secret data should remain protected. Two distrusting parties can be sure that a successful protocol execution results in the correct output and that a party's input can not be used for any other computation without said party's involvement.

In this report, active security is used to provide protection against an adversary that can have full hardware and software access to one of the parties.

It is impossible for a 2PC protocol to protect secret data when both parties are compromised. Thus, we need to employ organizational measures to reduce the risk of both parties being controlled by the adversary. These can include: code auditing and independent compiling, using separate cloud providers, and using public key infrastructure for authentication and securing communication channels.

The Sharemind MPC [1] framework provides secure two and three-party computation in the presence of a passive adversary along with many of the aforementioned organizational security measures. The aim of this report is to survey the literature and map out the approach for extending Sharemind MPC with two-party actively secure computation.

Contents of the report The report gives an overview of techniques for actively secure 2-party computation. Chapter 2 explains the concept of authenticated secret sharing in brief. In chapter 3, different approaches for the precomputation of correlated random values are detailed. Chapter 4 briefly covers garbled circuits in an active security setting. Lastly, chapter 5 lists some of the requirements for secure application development and compares them to available 2PC frameworks.

The report assumes some familiarity with multi-party computation and secret sharing.

2 Authenticated Secret Sharing

The most prominent technique for secure function evaluation in data intensive applications is secret sharing. The function is represented as a circuit consisting of gates and wires. The circuit is evaluated gate by gate with an interactive protocol and the input and output wires of the gates hold secret shared values.

In the Sharemind MPC framework, functions for secure evaluation are written in the SecreC language [2]. SecreC code is then compiled into a bytecode that is similar to a circuit representation. The parties hosting the Sharemind MPC server can then securely evaluate the bytecode with secure protocols for each bytecode instruction.

In the 2-party setting, we need a protocol that is secure against a dishonest majority, that is, one of the two parties can be corrupted by an adversary. Because of the dishonest majority requirement, many protocols based on compiling passively secure protocols into actively secure ones such as [3, 4] are ruled out. Protocols such as GMW [5] which require expensive zero-knowledge proofs to achieve active security are too inefficient for large data volumes. The SPDZ family of protocols [6, 7] are currently the best approach for dishonest majority active security.

Additive secret sharing is used in many passively secure MPC protocols. To achieve active security, the parties need to be able to check that the other party correctly evaluated each gate. In the SPDZ protocol, a passively secure protocol is enhanced with homomorphic message authentication codes to allow a party to check the other's work without revealing their secrets.

In Sharemind MPC, integer data types are supported. We will focus on the SPDZ2k protocol [7] because it allows for evaluating functions on finite rings such as 64-bit integers.

Definition 1 (Message authentication code) *A message authentication system consists of three algorithms (G, A, V). Algorithm G outputs a key k. Algorithm A takes as input a message m and a key k and outputs a message authentication code $s = A_k(m)$. Algorithm V takes as input the message authentication code s, the key k and the message m and outputs $accept$ if and only if $s = A_k(m)$ and $reject$ otherwise.*

The authenticated secret sharing scheme consists of additive shares of the input value x and shares of a message authentication code $m = x \cdot \alpha$. The key α is secret and both parties hold an additive share of the key.

Definition 2 (Authenticated secret sharing) *An authenticated secret share of $x \in \mathbb{Z}_{2^k}$ is the tuple (x_i, m_i^x, α_i) where $i \in \{1, 2\}$ and*

- $x_1 + x_2 = x' \mod 2^{k+s}$,
- $x' = x \mod 2^k$,
- $m^x = m_1^x + m_2^x \mod 2^{k+s}$,
- $\alpha = \alpha_1 + \alpha_2 \mod 2^{k+s}$,
- $m^x = x' \cdot \alpha \mod 2^{k+s}$.

The MAC m is created under a global key α and both parties hold additive shares of the MAC and the key. We use $[\![x]\!]$ to denote the set of authenticated secret shares of x.

A malicious adversary who wants to alter the value of x has to also alter the MAC share so that the MAC relation holds. This is equivalent to the adversary guessing the s least significant bits of α. Thus, the s additional bits in the ring form the security parameter. Setting $s \geq 40$ is most common.

2.1 Circuit Evaluation

At the start of evaluation, the parties sample random MAC key shares α_1 and α_2 and share their secret inputs x and y such that P_1 holds $(x_1, m_1^x, \alpha_1), (y_1, m_1^y, \alpha_1)$ and P_2 holds $(x_2, m_2^x, \alpha_2), (y_2, m_2^y, \alpha_1)$. These shares are then used to evaluate the gates in the circuit computing f. Subsequently, we implicitly assume that all arithmetic on shares is $\mod 2^{k+s}$.

Linear gates such as addition, subtraction and multiplication by a non-secret constant can be computed without interaction between the two parties:

$$[\![x]\!] + [\![y]\!] = \{(x_1 + y_1, m_1^x + m_1^y, \alpha_1), (x_2 + y_2, m_2^x + m_2^y, \alpha_2)\}$$

$$c \cdot [\![x]\!] = \{(c \cdot x_1, c \cdot m_1, \alpha_1), (c \cdot x_2, c \cdot m_2, \alpha_2)\}$$

$$c + [\![x]\!] = \{(c + x_1, c\alpha_1 + m_1, \alpha_1), (x_2, c\alpha_2 + m_2, \alpha_2)\}.$$

Before looking at the protocol for evaluating multiplication gates, we will look at how to open a secret shared value and check the correctness of the MAC.

The procedure for checking the MAC requires committing to a secret value before revealing it. A commitment scheme [8] is used to ensure that a party can not change their mind about a value after learning some additional information. This eliminates the advantage of receiving the other party's value before sending your own.

Using the commitment scheme, we can define the procedure MACCheck in Figure 1 for opening a single shared value and checking the MAC.

Procedure MACCheck

Procedure for opening a value $[\![x]\!]$ and checking the MAC.

1. The parties obtain a shared random value $[\![r]\!]$.
2. The parties compute $[\![y]\!] = [\![x]\!] + 2^k \cdot [\![r]\!]$ locally.
3. Both parties send their share y_i to the other party and reconstruct $y = y_1 + y_2 \mod 2^k$
4. Both parties commit to $z_i = m_i^y - y \cdot \alpha^i$ where m_i^y is a MAC share of $[\![y]\!]$
5. All parties open their commitments and check $z = z_1 + z_2 = 0 \mod 2^{k+s}$
6. If the check passes, output y, otherwise abort.

Figure 1. Procedure for opening a shared value and checking the MAC.

In step 2, the procedure masks the top s bits of x with a random value. This is done to prevent leakage of whether x has overflowed.

The MACCheck procedure is used whenever a private value is opened to check that the MAC relation holds. If the relation does not hold, than the parties have performed inconsistent operations on the private value. When a party detects such an inconsistency, it should halt or *abort* the protocol execution.

A vector of MACs can be opened in a more efficient manner by checking a random linear combination of MACs at once.

Procedure BatchMACCheck

Procedure for opening a values $[\![x_1]\!], \dots, [\![x_t]\!]$ and checking the MAC.
Opening the shared values: for every $i \in 1, \dots, t$:

1. The parties obtain a shared random value $[\![r_i]\!]$.
2. The parties compute $[\![y_i]\!] = [\![x_i]\!] + 2^k \cdot [\![r_i]\!]$ locally.
3. Both parties send their share of $[\![y_i]\!]$ to the other party and reconstruct $y_i = y_{i\,1} + y_{i\,2} \mod 2^k$

Checking the MACs:

1. Both parties sample random $\chi_1, \dots, \chi_t$.
2. Both parties compute
$$z_j = \sum_{i=1}^{t} \chi_i \cdot m_j^{y_i} - \chi_i \cdot y_i \cdot \alpha_j$$
where $m_j^{y_i}$ is a MAC share of $[\![y_i]\!]$ and $j \in \{0, 1\}$.
3. The parties commit to z_j.
4. The parties open their commitments and check $z = z_1 + z_2 = 0 \mod 2^{k+s}$
5. If the check passes, output y, otherwise abort.

Figure 2. Procedure for opening a vector of shared values and checking the MACs.

Using the procedure for opening secret values and checking MACs, we can define the procedure for multiplying two secret values. The multiplication gate in SPDZ2k requires pre-generated correlated randomness in the form of multiplication triples $([\![a]\!], [\![b]\!], [\![c]\!])$ where $c = a \cdot b$ and a and b are random. We will look at methods for generating these triples in chapter 3.

Procedure Multiplication

Procedure for multiplying values $[\![x]\!]$ and $[\![y]\!]$.

1. The parties obtain a random multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$.
2. Use **MACCheck** to open and check $[\![x]\!] - [\![a]\!]$ as ϵ and $[\![y]\!] - [\![b]\!]$ as δ.
3. Locally compute $[\![x \cdot y]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \delta \cdot [\![a]\!] + \epsilon \cdot \delta$.

Figure 3. Procedure for multiplying secret shared values

The final component of the online phase that we need to evaluate a circuit is input sharing and authenticating. This can be done using a pre-generated shared random value called the input mask.

With these components we can assemble the online phase of the SPDZ2k protocol for function evaluation.

The simple computations and low communication make the online phase of SDPZ2k very efficient, one multiplication requires sending 4 ring elements in two rounds. With a latency of 0.2 ms, the SPDZ2k protocol in mp-spdz [9] has an amortized throughput of 760 000 multiplications

Procedure Input sharing

Procedure for secret sharing input x from party P_i.

1. The parties obtain a shared random value $[\![r]\!]$ where r is known to P_i.
2. P_i sends $\epsilon = x - r$ to the other party.
3. The parties set $[\![x]\!] = [\![r]\!] + \epsilon$.

Figure 4. Procedure for creating an authenticated secret sharing of a party's input

Protocol SPDZ2k online phase

The parties securely evaluate the function f with inputs x and y from P_1 and P_2 respectively.

1. The parties have generate MAC key shares $\alpha_i \xleftarrow{\$} \mathbb{Z}_{2^{k+s}}$, multiplication triples and input masks as part of the preprocessing phase.
2. The parties create authenticated secret sharings $[\![x]\!]$ and $[\![y]\!]$ using the previous procedure.
3. The parties evaluate the gates of the circuit computing f:
   - Linear gates are evaluated without interaction.
   - Multiplication gates are evaluated following the procedure **Multiplication**.
4. The output wires of the circuit are revealed with procedure **MACCheck**. Output the revealed values.

Figure 5. Online phase of SPDZ2k protocol for function evaluation.

per second. With a larger latency of 100ms, the amortized throughput is 175 000 multiplications per second. Most of the complexity of SPDZ2k is in the preprocessing phase.

The online phase requires authenticated shares of random values, multiplication triples and input sharing masks. Chapter 3 looks at various approaches to precomputing these values.

2.2 Binary Circuits and Domain Conversion

The authenticated secret sharing scheme thus far can only evaluate circuits with linear and multiplication gates. While all other functions can be expressed using these gates, doing so would be highly inefficient. Some functions are more naturally expressed as binary circuits, so conversions between binary and arithmetic sharings are desirable. Additionally, we would like to provide gates for non-linear functions such as truncation and comparison.

The SPDZ2k authenticated secret sharing scheme over $\mathbb{Z}_{2^k}$ can also be used for evaluating binary circuits in $\mathbb{Z}_2$. The drawback of this approach is that the shares of the secret 1-bit value and MAC must be in a larger ring $\mathbb{Z}_{2^{1+s}}$. Thus, the communication overhead of any binary operation is very large relative to the input size.

The SPDZ protocol over $\mathbb{Z}_p$ can be used instead but the soundness error is dependent on the size of the field. Thus, for $\mathbb{Z}_2$, the chance of a cheating party not being detected is $\frac{1}{2}$. More MACs can be used to reduce the soundness error. In [10], a protocol for authenticating binary shares and generating binary multiplication triples from correlated oblivious transfer extensions

(COTe) is given.

The main approach for converting between binary and arithmetic shares in a 2-party active protocol is to use another kind of correlated random values to mask the secret value [11, 12].

A *doubly authenticated bit* (daBit) is a pair $(\llbracket r \rrbracket, \llbracket r \rrbracket_2)$ where $r \in \{0, 1\}$ is uniformly random and $\llbracket r \rrbracket_2$ is an authenticated binary secret share of r. In other words, a daBit is a single bit value secret shared in two different ways. Using this pair, a binary shared value $\llbracket x \rrbracket_2$ can be converted to $\llbracket x \rrbracket$:

1. Open $d = \llbracket r \rrbracket_2 \oplus \llbracket x \rrbracket_2$
2. Compute locally $\llbracket x \rrbracket = d + \llbracket r \rrbracket - 2d \cdot \llbracket r \rrbracket$

Extended doubly authenticated bits (edaBits) is a tuple $(\llbracket r \rrbracket, \llbracket r_0 \rrbracket_2, \ldots, \llbracket r_k \rrbracket_2)$, where $r_0, \ldots, r_k$ is the bit decomposition of r. Converting a value $\llbracket x \rrbracket$ in an arithmetic domain $x \in \mathbb{Z}_{2^k}$ to a authenticated secret shared bit decomposition is easy using edaBits:

1. Open $d = \llbracket x \rrbracket - \llbracket r \rrbracket$
2. Compute $[x_0]_2 \ldots [x_k]_2 \leftarrow$ BinaryAdder$(c, \big[[r_0]_2, \ldots [r_k]_2\big])$

The binary adder circuit which consists of boolean gates and implements addition of k-bit numbers is implemented in the authenticated binary secret sharing scheme.

With edaBits, other non-linear gates can be implemented. In [12], protocols for bitshift and truncation are given. In [13], an efficient comparison protocol uses two edaBits. Protocols using edaBits are explained in simpler terms in [14].

3 Preprocessing

The protocol for function evaluation using authenticated secret sharing depends on a couple of different forms of correlated random values. These values are independent of the function input and can thus be generated ahead of time in a *preprocessing phase*.

The ideal goal is a precomputation protocol that is fast enough to not be a performance bottleneck in data intensive applications. Such a protocol could be executed along side the function evaluation protocol to produce correlated random values as they are needed.

In the following section, two approaches are illustrated: homomorphic encryption based and oblivious transfer based preprocessing.

3.1 Homomorphic Encryption Based Preprocessing

The original SPDZ protocol uses a somewhat-homomorphic encryption (SHE) scheme for preprocessing multiplication triples. Somewhat-homomorphic encryption allows for adding and multiplying ciphertexts so that the resulting ciphertext is an encryption of the sum or product respectively. The number of subsequent operations that can be performed on a ciphertext is limited by the depth threshold of the SHE scheme. The SPDZ preprocessing [6] requires a single multiplication and addition of ciphertexts.

To generate a multiplication triple $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ where $a \cdot b = c \mod p$, the parties generate random shares of a and b locally. They then use the homomorphic property of the SHE scheme to encrypt their shares and compute ciphertexts of the shares of $a \cdot b$. A corrupted party could generate incorrect ciphertexts resulting in incorrect triples that could be exploited in the online phase. A zero-knowledge proof of plaintext knowledge is necessary to assure the correctness of the ciphertext.

The SHE scheme is also used to generate the MACs for the shared triples to obtain authenticated secret sharings of the triples.

Finally, the multiplicative property of the triple must be checked. This is done through the Triple-Sacrifice procedure in Figure 6 where one triple is used to check the validity of another. The triple sacrifice step is a common method in many preprocessing protocols.

Procedure TripleSacrifice

Procedure for checking whether $\llbracket a \rrbracket \cdot \llbracket b \rrbracket = \llbracket c \rrbracket$ through sacrificing a potentially incorrect triple $(\llbracket f \rrbracket, \llbracket g \rrbracket, \llbracket h \rrbracket)$

1. The parties obtain two triples $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$, $(\llbracket f \rrbracket, \llbracket g \rrbracket, \llbracket h \rrbracket)$. The multiplicative property or MAC relation of these triples is not guaranteed to hold.
2. Optain and open a random authenticated secret shared value $\llbracket r \rrbracket$.
3. Open $p = r \cdot \llbracket a \rrbracket - \llbracket f \rrbracket$ and $s = \llbracket b \rrbracket - \llbracket g \rrbracket$.
4. Open $z = r \cdot \llbracket c \rrbracket - \llbracket h \rrbracket - s \cdot \llbracket f \rrbracket - p \cdot \llbracket g \rrbracket - s \cdot p$.
5. If $z = 0$, use the triple $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ in subsequent secure computation, otherwise discard the triple and abort.

Figure 6. Procedure for checking the multiplicative property of a triple.

All combined, the preprocessing phase of SPDZ is very slow due to the high communication cost and expensive somewhat-homomorphic encryption. The paper states that preprocessing a single multiplication triple takes 13ms in a 3-party setting [6].

Later advancements in SPDZ preprocessing achieve higher triple throughput by using an only additively homomorphic encryption scheme. The Overdrive protocol [15] achieves 59000 triples per second in a 64-bit prime field. In $\mod 2^k$, the MonZa protocol [16] achieves just 19 triples per second.

Since the current state of the art of homomorphic encryption based preprocessing is a major bottleneck for secure function evaluation, we will consider an oblivious transfer based approach instead.

3.2 Oblivious Transfer Based Preprocessing

Oblivious transfer (OT) is a two party protocol between a sender and a receiver. In a 1-out-of-2 OT, the sender has two messages m_0 and m_1 and the receiver obliviously chooses one of the messages. The sender does not learn the receivers choice and the receiver does not learn the other message. The functionality of OT is illustrated in Figure 7. Very efficient constructions exist for computing a large number of oblivious transfers with both active and passive security [17, 18].

The main reason for basing preprocessing protocols on oblivious transfer variants is that efficient constructions exist for pseudorandom correlation generators (PCGs) that create large quantities of random OTs from a small setup phase. Using PCGs, the number of communication rounds and communication amount can be reduced enough that the preprocessing can be done on the fly during secure function evaluation [19].

Oblivious transfer and variants of it can be used for generating multiplication triples for evaluating multiplication gates in the authenticated secret sharing scheme. Additionally, OT variants can be used to authenticate secret shared values by computing the corresponding MAC shares.

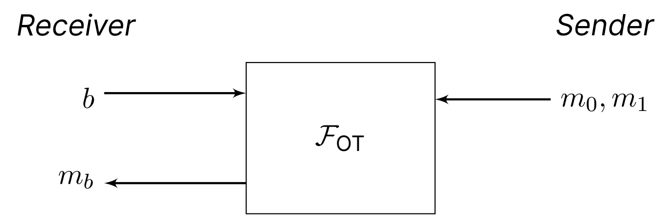

Figure 7. Oblivious transfer functionality.

Authenticating Shared Values We can use OT based techniques to create MAC shares for secret shared values. The parties start of with shares α_1 and α_2 of the MAC key and shares x_1 and x_2 of a secret value. The goal of authentication is to get shares m_1 and m_2 of the MAC such that $m_1 + m_2 = (\alpha_1 + \alpha_2)(x_1 + x_2) \mod 2^k$.

A variant of OT called correlated oblivious product evaluation (COPE) where the sender inputs x, the receiver inputs Δ and the parties learn q and t such that $q + t = x \cdot \Delta$, is used to authenticate a secret sharing in the MASCOT protocol [20].

The MASCOT protocol for authenticating shares in $\mathbb{Z}_p$ is secure against a malicious adversary and dishonest majority but it is based on a passively secure OT extension protocol. A privacy amplification technique is used to achieve malicious security which is of interest in the $\mathbb{Z}_{2^k}$ case as well.

The SPDZ2k protocol uses a similar arithmetic OT variant to authenticate values in $\mathbb{Z}_{2^k}$.

Oblivious linear evaluation (OLE) is a common variant of OT where a sender, with coefficients a and b, and a receiver with input x, compute $ax + b$ while keeping their inputs secret. The output $ax+b$ is only seen by the receiver. Random oblivious linear evaluation (R-OLE) is a version of OLE where the inputs x, a, and b are randomly sampled. The functionality of R-OLE is illustrated in Figure 8. R-OLE is of particular interest because it can be used to cheaply implement regular OLE and R-OLE output pairs can be independently generated without communication using silent-OT [18].

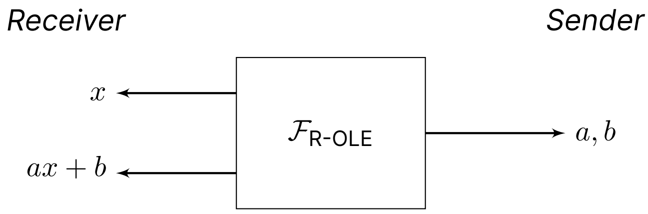

Figure 8. Random oblivious linear evaluation functionality.

The OLE based authentication procedure is described in Figure 9.

Procedure Authenticate

Procedure for authenticating a secret sharing of $x = x_1 + x_2 \mod 2^k$. The parties hold MAC key shares α_1 and α_2. The parties want to obtain MAC shares m_1 and m_2 such that $m_1 + m_2 = (\alpha_1 + \alpha_2)(x_1 + x_2) \mod 2^k$.

---

1. P_1 samples a random r_1 and P_2 samples a random r_2.
2. The parties run $\mathcal{F}_{\text{OLE}}$ with P_1 as the sender. P_1 inputs α_1 and r_1, P_2 receives $t_1 = \alpha_1 x_2 + r_1$.
3. The parties run $\mathcal{F}_{\text{OLE}}$ with P_2 as the sender. P_2 inputs α_2 and r_2, P_1 receives $t_2 = \alpha_2 x_1 + r_2$.
4. P_1 sets $m_1 = \alpha_1 x_1 + t_2 - r_1$ and P_2 sets $m_2 = \alpha_2 x_2 + t_1 - r_2$.

Figure 9. Procedure for authenticating a secret sharing.

Using the authentication procedure, the parties can create input masks $[\![r]\!]$ where r is known to one of the parties. The parties just sample random shares of r and use the authentication procedure to obtain $[\![r]\!]$. Then $[\![r]\!]$ is opened to one of the parties.

Multiplication Triples from OLE The output of R-OLE can be interpreted as a multiplication triple:

1. The parties P_1 and P_2 call $\mathcal{F}_{\text{R-OLE}}$ twice. P_1 receives x, x', y, y' and P_2 receives a, b, a', b' such that $y = ax + b$ and $y' = a'x' + b'$.
2. Set $[u] := (x, a')$, $[v] := (x', a)$, and $[w] := (x'x + y + y', a'a - b - b')$.
3. $u \cdot v = (x + a') \cdot (x' + a) = x'x + xa + x'a' + a'a = x'x + y - b + y' - b' + x'a' + a'a = w$

These triples from R-OLE are not authenticated, to authenticate them the OLE based procedure can be used. The resulting authenticated triples might not be multiplicative if one party deviated. To verify that the multiplicative property holds, the sacrifice step from SPDZ can be used. This entails generating twice as many triples as necessary and sacrificing half of them to check the other half.

3.3 Pseudorandom Correlation Generators

The OT based preprocessing requires many calls to OLE and R-OLE functionalities. This approach to preprocessing is only efficient if these calls can be cheaply implemented. We will look at pseudorandom correlation generators (PCGs) for silent OT and OLE extension.

PCGs allow two parties to create a large amount of correlated random values without interaction after a small interactive setup phase. In the setup phase the parties run a protocol to generate a seed that they can then independently expand into many correlated random values. PCGs exist for many correlations such as random OT, random OLE and VOLE, and multiplication triples. The most efficient constructions are for OT and VOLE which allow the parties to generate millions of values in a second from a seed that is a couple of megabytes [19, 21, 22].

Definition 3 (PCG) *A pseudorandom correlation generator for a correlation $\mathcal{C}$ is a pair of algorithms (G, E) where $G(1^\kappa)$ outputs a pair of seeds (k_0, k_1) and $E(i, k_i)$ for $i \in \{0, 1\}$ outputs a bit string $R_i \in \{0, 1\}^n$ such that (R_0, R_1) is indistinguishable from $\mathcal{C}(1^\kappa)$.*

The starting point for constructing PCGs is *function secret sharing* (FSS) and the *learning parity with noise* (LPN) assumption [23]. With FSS, the parties split the function into hiding additive shares that can be independently evaluated. More concretely, given a function f in some class of functions, the parties generate keys k_0, k_1 and using a key k_b and input x, a party can evaluate y_b such that $y_0 + y_1 = f(x)$.

For constructing PCGs, we are interested in FSS for a special class of functions called point functions, $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(x) = 0$ for any $x \neq \alpha$. An FSS scheme that evaluates a point function is called a distributed point function (DPF). Multiple DPFs can be used to evaluate a multi-point function.

Definition 4 (MPFSS) *Let S be an ordered size t subset of $\{0, \ldots, n-1\}$. An (n, t)-multi-point function secret scheme is a pair of algorithms (G, E) where $G(S, \overrightarrow{y})$ outputs a pair of keys (K_0, K_1) such that $E(K_i)$ is a vector of additive secret shares of $\{s_i \mid s_i = \overrightarrow{y}_i$ if $i \in S$ otherwise $s_i = 0\}$.*

Definition 5 (Learning Parity with Noise (LPN) assumption) *Let $\mathcal{D}$ be a family of distributions over a ring $\mathcal{R}$ and $\mathcal{C}$ be a probabilistic code generation algorithm such that $\mathcal{C}(k, q, \mathcal{R})$ outputs a matrix $A \in \mathcal{R}^{k \times q}$. For dimension k, number of samples q and ring $\mathcal{R}$, the $(\mathcal{D}, C, \mathcal{R}) - \text{LPN}(k, q)$ assumption states that $\{(A, s \cdot A + e) \mid A \leftarrow C(k, q, \mathcal{R}), e \leftarrow \mathcal{D}_{k,q}, s \leftarrow \mathcal{R}^k\}$ is computationally indistinguishable from $\{(A, b) \mid A \leftarrow C(k, q, \mathcal{R}), b \leftarrow \mathcal{R}^q\}$.*

More informally, the LPN assumption states that for some code generator matrices A, a noisy codeword $s \cdot A + e$ from a uniformly random input s, is uniformly random. Different variants of the LPN assumption exist which result in more efficient PCGs [24].

A pseudorandom correlation generator for VOLE correlations is introduced in [22]. An outline of this PCG is given in Figure 10.

Procedure PCG for VOLE

$H_{m,n} \in \mathbb{F}^{m \times n}$ is a public parity check matrix of a dual code and t is the noise threshold. (G, E) is a (m, t)-multi-point function secret sharing scheme.

Setup

1. Pick a random size t subset $S \subset \{1, \ldots, m\}$ and a random vector $\vec{y} \in \mathbb{F}^t$ and random $x \in \mathbb{F}$
2. Compute $(K_0, K_1) = G(S, x \cdot \vec{y})$.
3. Set $seed_0 = (\mathbb{F}, m, n, K_0, S, \vec{y})$ and $seed_1 = (\mathbb{F}, m, n, K_1, x)$. Output the seeds to the respective parties.

Expand

1. Party P_0 gets $(\mathbb{F}, m, n, K_0, S, \vec{y})$. P_0 sets $\vec{\mu} = \text{spread}_n(S, \vec{y})$ and $\vec{v_0} = E(K_0)$. P_0 outputs $(\vec{\mu} \cdot H_{m,n}, -\vec{v_0} \cdot H_{m,n})$.
2. Party P_1 gets $(\mathbb{F}, m, n, K_1, x)$. P_1 sets $\vec{v_1} = E(K_1)$ and outputs $(x, \vec{v_1} \cdot H_{m,n})$.

Figure 10. Outline of a procedure for generating and expanding keys of a pseudorandom correlation generator for vector oblivious linear evaluation.

The correctness of this PCG follows from the correctness of the multi-point function secret sharing. The security of the PCG holds if the MPFSS is secure and dual LPN assumption holds for the chosen parameters.

A maliciously secure PCG for VOLE correlations over an integer ring is used for zero-knowledge proofs in [25]. A PCG for the daBits and edaBits is given in [26]. Assuming the hardness of decoding some LDPC codes instead of the LPN assumption results in more efficient PCGs for VOLE and OT [21]. The ring-LPN assumption can be used to construct PCGs for OLE correlations [24].

PCGs can be used to reduce the bottleneck of the preprocessing phase in two-party computation. Instead of a homomorphic encryption based preprocessing for multiplication triples and other correlated randomness, a small setup phase generates keys for a number of different PCGs that are expanded during the online phase. The remaining online phase is *non-cryptographic* and computationally much less complex. The resulting protocol steps are illustrated in Figure 11.

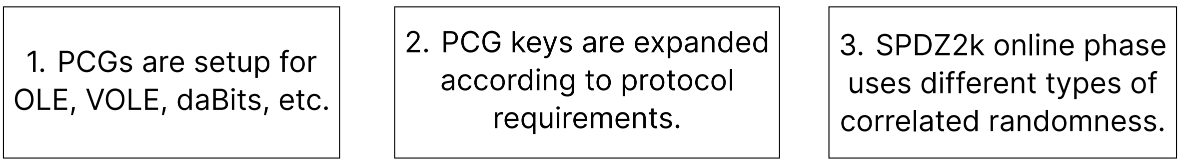

Figure 11. Outline of a SPDZ2k style secure function evaluation protocol using PCGs.

4 Garbled Circuits

Garbled circuits (GC) is an approach for secure function evaluation that has a constant communication pattern, independent of the function circuit. Instead of evaluating each gate in a circuit with a secure protocol, garbled circuits transforms the circuit into a privacy preserving garbled circuit that can be evaluated by one of the parties locally.

The protocol works in two phases. Firstly, one of the parties, called the *garbler*, embeds their input into the circuit and creates ”encrypted” truth-tables for the gates in the circuit. The garbled circuit is then sent to the other party, the *evaluator*, who uses their input to evaluate the circuit to receive an ”encrypted” result which can then be opened by the garbler.

The garbling and evaluating steps are computationally heavy and the total communication cost is higher than that of secret sharing based function evaluation, but unlike secret sharing, the number of communication rounds is constant and low. Therefore, for very deep circuits that have relatively small inputs, GC based function evaluation can be faster.

Circuit Garbling Let $G(A, B, i)$ be a pseudo-random function where A and B are k-bit keys, $i \in \{0, \dots, T\}$, and the output is $2k$ bits. In loose terms, the circuit garbling works as follows:

1. For each wire $i \in \{0, \dots, T\}$ in the circuit, the garbler picks two random k-bit keys (K_0^i, K_1^i).
2. Let i be the index of an output wire of an AND gate and l and r be its left and right input wires respectively. To create the garbled tabled of the gate:
   a. For each $(a, b) \in \{0,1\} \times \{0,1\}$, $C_{a,b} = G(K_a^l, K_b^r, i) \oplus (K_{a \wedge b}^i \parallel 0^k)$.
   b. Randomly permute $(C_{00}, C_{01}, C_{10}, C_{11})$
3. Any other logical gate can be garbled in the same manner.

In the beginning of evaluation, the evaluator has inputs $b_0, \dots, b_n$ for wires $i \in \{0, \dots, n\}$. Using an oblivious transfer protocol, the evaluator obtains keys $K_{b_0}^0, \dots, K_{b_n}^n$ from the garbler.

The evaluator uses the input wire keys to evaluate the circuit gate-by-gate. For each gate i, the garbler computes $G(K_x^l, K_y^r, i)$, and uses it to decrypt the garbled table. Only one of the decryptions will be correct giving the value $(K_{x \wedge y}^i \parallel 0^k)$.

More advanced constructions use key derivation algorithms to reduce the communication of wire keys and XOR gates can be evaluated without the need for a garbled table [27].

Actively Secure Garbled Circuits An actively corrupted garbler can break the evaluator's privacy by garbling a circuit that reveals the evaluator's secret input. Since the evaluator can not see the logic of the underlying circuit from the garbled circuit, the garbler has many ways to cheat. A way to mitigate this is to garble lots of circuits so that the evaluator can open a random selection of them and validate the function that they compute. This approach is referred to as cut-and-choose [28].

A newer technique for actively secure garbled circuits uses a preprocessing phase and correlated randomness to create an *authenticated garbled circuit* [29]. The technique is comparable to the MACs in the authenticated secret sharing scheme. The resulting protocol evaluates AES circuits in 6.7 ms when amortized over 1048 executions.

In brief, the authenticated garbled circuit is secret shared between the garbler and the evaluator. As a result, the garbling step is performed in a distributed manner and the garbler can not change the logic of the garbled circuit on their own.

An effective approach is to combine authenticated secret sharing and garbled circuits in a mixed protocol. The SCALE-MAMBA MPC framework converts between authenticated secret shares and garbled circuit representation using edaBits [30]. Evaluating neural networks is more efficient with a mixed protocol where neuron activation functions are evaluated using garbled circuits and matrix multiplications are done using secret sharing [31].

5 2PC Frameworks for Application Development

The most notable frameworks for developing applications using two-party actively secure computation are MP-SPDZ [9] and SCALE-MAMBA [32]. Both are intended to be research tools for developing and benchmarking MPC protocols and applications. Their standout feature is the large number of deployment configurations and underlying MPC protocols supported.

MP-SPDZ implements, among others, SPDZ2k online phase with homomorphic encryption and oblivious transfer based preprocessing. The protocol includes arithmetic functions for integers, floating point numbers and fixed point numbers. Applications on top of MP-SPDZ are programmed in a Python-like programming language.

SCALE-MAMBA implements SPDZ with homomorphic encryption based preprocessing for the two-party case. Arithmetic functions for integers, floating point numbers and fixed point numbers are included. In recent versions of SCALE-MAMBA, MPC applications are built in the Rust language.

In both frameworks, branching over a public conditional and array types are supported. MP-SPDZ additionally has built-in ORAM support and neural network evaluation.

The major deficit of these frameworks is the lack of persistent secure table storage and table operations. For application deployment, another important missing feature is role assignment and access control to indicate which party is allowed to execute which secure computation. These are features that are available and commonly used in MPC applications built using Sharemind MPC.

Adding actively secure two-party computation to Sharemind MPC Sharemind MPC includes a large library of functions for three-party passively secure MPC. These functions are implemented in a domain specific language for MPC protocols (PDSL) [2]. PDSL is a side-effect free functional language. The PDSL compiler vectorises protocols and applies various optimizations as well as checking the privacy of the resulting protocols [33].

For implementing actively secure two-party protocols the PDSL compiler has to be adapted. Active security adds the side effect of protocol abort. Thus, the language has to account for this new possible outcome. A failed security check, such as a MAC check or commitment opening, should result in the party aborting the remaining computation and not sending any additional messages to the other party. Since the Sharemind MPC application server can execute multiple secure computations simultaneously, one aborted computation should not interfere with other computations.

The passively secure three party protocols do not rely on any precomputed correlated randomness, only a regular tape of pseudo-random bits. The compiler accounts for all consumed random bits and for every protocol specifies the length of the random tape. For 2PC, the compiler has to be adapted to account for the many different types of correlated pseudo-random values.

The static privacy checking of [33] does not apply to actively secure two-party protocols and how to achieve a similar static checker is not immediately clear.

Bibliography

[1] Dan Bogdanov, Sven Laur, and Jan Willemson. "Sharemind: A Framework for Fast Privacy-Preserving Computations". In: *Computer Security - ESORICS 2008*. Ed. by Sushil Jajodia and Javier Lopez. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 192–206. ISBN: 978-3-540-88313-5.

[2] Jaak Randmets. "Programming Languages for Secure Multi-party Computation Application Development". PhD thesis. University of Tartu, 2017. URL: http://hdl.handle.net/10062/56298.

[3] Ivan Damgård, Claudio Orlandi, and Mark Simkin. *Yet Another Compiler for Active Security or: Efficient MPC Over Arbitrary Rings*. Cryptology ePrint Archive, Paper 2017/908. https://eprint.iacr.org/2017/908. 2017. URL: https://eprint.iacr.org/2017/908.

[4] Hendrik Eerikson et al. "Use Your Brain! Arithmetic 3PC for Any Modulus with Active Security". In: *1st Conference on Information-Theoretic Cryptography (ITC 2020)*. Ed. by Yael Tauman Kalai, Adam D. Smith, and Daniel Wichs. Vol. 163. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 5:1–5:24. ISBN: 978-3-95977-151-1. DOI: 10.4230/LIPIcs.ITC.2020.5. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12110.

[5] O. Goldreich, S. Micali, and A. Wigderson. "How to Play ANY Mental Game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 218–229. ISBN: 0897912217. DOI: 10.1145/28395.28420. URL: https://doi.org/10.1145/28395.28420.

[6] Ivan Damgård et al. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662. ISBN: 978-3-642-32009-5.

[7] Ronald Cramer et al. "SPDZ2k: Efficient MPC mod 2k for Dishonest Majority". In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Cham: Springer International Publishing, 2018, pp. 769–798. ISBN: 978-3-319-96881-0.

[8] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer, Jan. 2010. ISBN: 978-3-642-14302-1. DOI: 10.1007/978-3-642-14303-8.

[9] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020. DOI: 10.1145/3372297.3417872. URL: https://doi.org/10.1145/3372297.3417872.

[10] Tore Kasper Frederiksen et al. *A Unified Approach to MPC with Preprocessing using OT*. Cryptology ePrint Archive, Paper 2015/901. https://eprint.iacr.org/2015/901. 2015. URL: https://eprint.iacr.org/2015/901.

[11] Dragos Rotaru and Tim Wood. *MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security*. Cryptology ePrint Archive, Paper 2019/207. https://eprint.iacr.org/2019/207. 2019. URL: https://eprint.iacr.org/2019/207.

[12] Daniel Escudero et al. *Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits*. Cryptology ePrint Archive, Paper 2020/338. https://eprint.iacr.org/2020/338. 2020. URL: https://eprint.iacr.org/2020/338.

[13] Eleftheria Makri et al. *Rabbit: Efficient Comparison for Secure Multi-Party Computation*. Cryptology ePrint Archive, Paper 2021/119. https://eprint.iacr.org/2021/119. 2021. URL: https://eprint.iacr.org/2021/119.

[14] Daniel Escudero. "Multiparty Computation over Z/2kZ". PhD thesis. Aarhus University, 2021.

[15] Marcel Keller, Valerio Pastro, and Dragos Rotaru. "Overdrive: Making SPDZ Great Again". In: *Advances in Cryptology – EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Cham: Springer International Publishing, 2018, pp. 158–189. ISBN: 978-3-319-78372-7.

[16] Dario Catalano et al. *MonZa: Fast Maliciously Secure Two Party Computation on Z2k*. Cryptology ePrint Archive, Report 2019/211. https://ia.cr/2019/211. 2019.

[17] Marcel Keller, Emmanuela Orsini, and Peter Scholl. *Actively Secure OT Extension with Optimal Overhead*. Cryptology ePrint Archive, Paper 2015/546. https://eprint.iacr.org/2015/546. 2015. URL: https://eprint.iacr.org/2015/546.

[18] Tung Chou and Claudio Orlandi. *The Simplest Protocol for Oblivious Transfer*. Cryptology ePrint Archive, Paper 2015/267. https://eprint.iacr.org/2015/267. 2015. URL: https://eprint.iacr.org/2015/267.

[19] Damiano Abram et al. *An Algebraic Framework for Silent Preprocessing with Trustless Setup and Active Security*. Cryptology ePrint Archive, Paper 2022/363. https://eprint.iacr.org/2022/363. 2022. URL: https://eprint.iacr.org/2022/363.

[20] Marcel Keller, Emmanuela Orsini, and Peter Scholl. *MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer*. Cryptology ePrint Archive, Paper 2016/505. https://eprint.iacr.org/2016/505. 2016. DOI: 10.1145/2976749.2978357. URL: https://eprint.iacr.org/2016/505.

[21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. *Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes*. Cryptology ePrint Archive, Paper 2021/1150. https://eprint.iacr.org/2021/1150. 2021. URL: https://eprint.iacr.org/2021/1150.

[22] Elette Boyle et al. *Compressing Vector OLE*. Cryptology ePrint Archive, Paper 2019/273. https://eprint.iacr.org/2019/273. 2019. DOI: 10.1145/3243734.3243868. URL: https://eprint.iacr.org/2019/273.

[23] Elette Boyle et al. *Efficient Pseudorandom Correlation Generators: Silent OT Extension and More*. Cryptology ePrint Archive, Paper 2019/448. https://eprint.iacr.org/2019/448. 2019. URL: https://eprint.iacr.org/2019/448.

[24] Elette Boyle et al. *Efficient Pseudorandom Correlation Generators from Ring-LPN*. Cryptology ePrint Archive, Paper 2022/1035. https://eprint.iacr.org/2022/1035. 2022. DOI: 10.1007/978-3-030-56880-1_14. URL: https://eprint.iacr.org/2022/1035.

[25] Carsten Baum et al. *Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs Over $\mathbb{Z}_{2^k}$*. Cryptology ePrint Archive, Paper 2022/819. https://eprint.iacr.org/2022/819. 2022. URL: https://eprint.iacr.org/2022/819.

[26] Sameer Wagh. *BarnOwl: Secure Comparisons using Silent Pseudorandom Correlation Generators*. Cryptology ePrint Archive, Paper 2022/800. https://eprint.iacr.org/2022/800. 2022. URL: https://eprint.iacr.org/2022/800.

[27] Vladimir Kolesnikov and Thomas Schneider. “Improved Garbled Circuit: Free XOR Gates and Applications”. In: *Automata, Languages and Programming*. Ed. by Luca Aceto et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 486–498.

[28] Yan Huang, Jonathan Katz, and Dave Evans. *Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose*. Cryptology ePrint Archive, Paper 2013/081. https://eprint.iacr.org/2013/081. 2013. URL: https://eprint.iacr.org/2013/081.

[29] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. *Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation*. Cryptology ePrint Archive, Paper 2017/030. https://eprint.iacr.org/2017/030. 2017. URL: https://eprint.iacr.org/2017/030.

[30] Abdelrahaman Aly et al. *Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE*. Cryptology ePrint Archive, Paper 2019/974. https://eprint.iacr.org/2019/974. 2019. URL: https://eprint.iacr.org/2019/974.

[31] Arpita Patra et al. “ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2165–2182. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/patra.

[32] Abdelrahaman Aly et al. *SCALE–MAMBA v1.14 : Documentation*. 2022. URL: https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation-SCALE.pdf.

[33] Martin Pettai and Peeter Laud. “Automatic Proofs of Privacy of Secure Multi-party Computation Protocols against Active Adversaries”. In: *2015 IEEE 28th Computer Security Foundations Symposium*. 2015, pp. 75–89. DOI: 10.1109/CSF.2015.13.