Secure Multi-Party Computation with a Helper*

Johannes Schneider

October 9, 2018

Abstract

A client wishes to outsource computation on confidential data to a network of parties. He does not trust a single party but believes that multiple parties do not collude. To solve this problem, we use the idea of treating one of the parties as a helper. A helper assists computation only. Often using more parties ensures confidentiality despite more corrupted parties. This does not hold for adding a helper. But a helper can in some cases lower the amount of communication asymptotically to the theoretical minimum of one bit per AND gate, improving significantly on schemes without a helper. It can also allow for very efficient computations of certain functions, as we show for the exponential function with public base.

1 Introduction

Cloud computing is on the rise with security remaining as one of the key challenges. A cloud provider must be fully trusted to refrain from misusing any confidential information. Often a single vulnerability, e.g. a corrupt system administrator, can put the confidentiality of a large amount of data at risk. Secure multi-party computation (MPC) is a computationally efficient approach that allows a client to outsource computation to a group of parties (or cloud providers), assuring that the client's information is prevented from misuse even if some parties are corrupted or cannot fully be trusted. Interaction is illustrated in Figure 1, where a client uses various cloud providers and one of them serves as a helper to facilitate computation. Dedicating one party for a special purpose, i.e. as a helper, is in contrast to existing schemes [20, 5, 1, 4, 8, 9]. Typically shares of a secret are distributed equally among all



Figure 1: A client outsources computation believing that no two cloud providers are dishonest. The right most provider serves as helper.

parties, and all parties behave identically, i.e. they perform the same computations but on different values. It is not clear, whether there is any benefit in deviating from this well-established body of work. Thus, the question we seek to answer is:

What are the advantages and disadvantages of using a helper, i.e. one party for assisting computation?

To this end, we develop techniques for MPC computation using a helper and compare them to the state-of-theart in MPC focusing on the three party case. One might conjecture that the use of a helper does not help in improving security, since it does not hold a share of a secret. This is indeed one of our findings. One might also conjecture that the minimum number of rounds for any non-trivial operation is at least two, since the helper must receive information and return information. In fact, several of our protocols match this (lower) bound. With respect to other classical metrics such as communication and local computation complexity, it seems harder to come up with reasonable conjectures.

^{*}This work extends the conference paper[27].

Paper	Transmitted Bits for		Rounds	Maximum
	AND of two single bits	AND of all pairs of v variables		Corrupted Parties
GMW '87[20]	>50	$> 3 \cdot v^2$	2	2
BMR '90[5]	>10	$> 3 \cdot v^2$	> 2	2
CCS '16 [1]	3	$3 \cdot v(v-1)/2$	1	2
JOS (This work)	5	v(v-1)/2 + 4v	2	1

Table 1: Comparison of three party protocols. XOR requires no communication in all schemes.

Our protocols using a helper are designed to minimize communication and round complexity, while keeping local computation complexity at the same level as the stateof-the-art. However, we also present a method involving a helper allowing to trade-off communication and round complexity for logical operations, i.e. large fan-in AND gates. Communication complexity has gained a lot in importance lately. All modern schemes for data analysis, such as Hadoop or Spark, rely typically on dataparallelism, i.e. performing the same computation for different parts of the data. Computation is done in a distributed fashion, which requires moving large amounts of data between computers (as is needed for MPC, in particular for big data). The bottleneck becomes bandwidth, i.e. the amount of information that can be transmitted, and not network latency, i.e. number of communication rounds. Since any Boolean circuit can be computed using basic building blocks such as 'AND' and 'XOR' gates, most MPC schemes focus on evaluating such circuits. Therefore, a natural question is:

How many bits must be transmitted using a helper to evaluate AND and XOR gates with non-trivial security guarantees?

Intuitively, this seems to be at least two bits per (AND) gate, since a helper must receive and transmit at least one bit. However, surprisingly, in case the same variables occur multiple times, we can reach the theoretical minimum of just one bit per gate as indicated in Table 1 with non-trivial security guarantees under the commonly found assumption that randomness is pre-shared among parties. This is almost a factor 3 less than prior work for three party computation. Unfortunately, this comes at a price that anyone striving for performance might well be willing to pay. Traditionally, the entire system is corrupted if any confidential information becomes available to any party. Honest, non-corrupted parties are

(implicitly) also assumed to abuse confidential information. We assume that the system is only compromised if a corrupted party, i.e. the attacker, gets secret information, but honest, non-corrupted parties participating in the computation would not abuse confidential information. Essentially, this means that an attacker can only corrupt one party in our scheme but two in other schemes as indicated in the last column of Table 1. Arguably the largest benefits of using a helper can be reaped when computing special functions such as the exponential function in this work, where mathematical laws involving two variables can be used to compute the function in just two rounds.

To summarize, a helper can imply a significant reduction of communication complexity in several situations under the condition that the system is only compromised if a corrupted party, i.e. the attacker, gets secret information.

- If the same variables occur in multiple operations so that the effort of sharing with the helper can be reduced.
- If the function to be computed allows leveraging the secret sharing principle of having encrypted values and keys.

1.1 Contributions

In summary, we make the following contributions:

We elaborate on the "distrust attacker" model assuming semi-honest parties and provide a mapping of security guarantees to the standard "distrust all" security model.

- We assess the usefulness of a helper assisting computation. We present a new scheme for MPC in the semi-honest model for boolean gates using three parties with non-trivial security guarantees. If variables occur multiple times, we improve on the amount of communication needed to compute multiple ANDs compared to prior work (see Table 1 and Related Work Section). We even reach an asymptotically optimal value of just one bit per AND gate if the number of evaluated AND gates is more than linear in the number of variables.
- We present a method to compute unbounded fan-in gates in constant rounds. It comes with a trade-off for communication and rounds. Using w variables and messages of size $O(w \cdot 2^{w-k})$ for arbitrary $k \in [2, w]$ an AND can be computed in $O(\log k)$ rounds and $O(w \cdot 2^{w-k})$ operations involving single bits.
- We present a statistically secure protocol for computing any exponential function a^x for a secret exponent x and public base a improving prior work (even in the same model) considerably.

1.2 Outline

After stating the model in Section 2, we provide an overview of the method and its motivation in Section 3, followed by describing the secret sharing in Section 4 and basic operations in Section 5, i.e. XOR and NOT. The basic ideas of the AND protocol are stated in Section 6 using four parties, i.e. two helpers. The number of helpers is reduced to one in Section 7. In Section 7.1 we derive a protocol that achieves amortized communication of just 1 bit per AND gate, if the same variables occur in a large number of gates. A trade-off between communication and round complexity for arbitrary fan-in gates is given in Section 7.3. A protocol for computing exponential functions with a public base is given in Section 7.2.

2 Model

We adopt the semi-honest model for client-server computation, where a curious but passive attacker can monitor a party completely, e.g. its memory, disc and CPU registers. A client holds an arbitrary amount of secret values. The

client wishes to evaluate a function using three parties, such that no corrupted party, i.e., attacker, can learn anything about the input or the output. The standard model is different since it is based on distrusting everybody. This means that no party should learn anything about the input. The standard, i.e. "distrust all", model assumes that honest, non-corrupted parties do not actively share any information with any other party throughout computation but can still not be trusted with any confidential information, i.e., they behave dishonestly as soon as they get secrets. In our model, an honest party would not misuse secret information, but an attacker would. We call this "distrust attacker" model. Any result for a "distrust all" model can be directly translated to the "distrust attacker" model:

Theorem 1. An MPC scheme remains confidentiality of secrets despite x corrupted parties for a "distrust all" model, if and only if it remains confidentiality despite x+1 corrupted parties in the "distrust attacker" model.

Proof. Assume that corrupted parties share their information with all parties. Assume that for a set of x+1 corrupted parties S in the distrust all model, at least one party A (not necessarily corrupted) can obtain some confidential information using the information from the parties S. Therefore, in the distrust attacker model, assume that $S \cup \{A\}$ are corrupted. In this case, all corrupted parties, i.e., also the attacker, obtain all information from the $S \cup \{A\}$ parties and, thus, also confidential information. Therefore, if an MPC schemes withstands x corrupted parties in the distrust all model it cannot withstand more than x+1 parties in the distrust attacker model.

Assume that for x+2 corrupted parties S in the distrust attacker model, at least one corrupted party $A \in S$ can obtain some confidential information. Therefore, in the distrust all model, assume that $S \setminus A$ are corrupted. Since A receives by assumption all information from parties $S \setminus A$ it obtains confidential information.

In all MPC schemes, typically, a client encrypts its secrets and distributes the shares among the parties, the parties compute the desired function and return their shares to the client. The client itself does not participate in the computation and, therefore, does not count as a party. This differs from the classical MPC model, where each party holds a secret (or at least a share), and the output should be

known by (at least) one party. We can emulate the classical model: The parties can always obtain the secret value of an output through collusion (rather than transmitting all their shares to the client). In case each party has a secret, each party can execute the same protocol for encryption and distribution of shares of its secret as a client having all secrets. Thus, the extension to using several clients (each having some secret value) is obvious.

Our simplest network consists of a client, a key holder (KH) and an encrypted value holder (EVH) and a helper. The client communicates with the KH and EVH. A network with three parties is shown in Figure 1. We assume perfectly secure communication channels between parties. We assume that there is pre-shared randomness among pairs of parties. This implies that keys do not have to be transmitted but can be regarded as pre-shared. In practice, one can generate keys using a shared secret seed and pseudo-random number generators.

3 Overview of Approach

Our scheme called JOS requires at least three parties, where each party can be thought of having a dedicated role: a keyholder (KH), an encrypted value holder (EVH) and a helper. Beyond three parties, the number of keyholders increases. Generally, the keyholder stores keys but it has no access to ciphertexts. The encrypted value holder stores encrypted values but no keys. Note, there are circumstances, where this distinction has to be seen less strict, e.g., there might be two encryptions of the same secret such that one is held by the KH. A helper assists computations. It might obtain keys and encrypted values that do not match, i.e. none of its keys can be used to decrypt any of its encrypted values. The helper should only facilitate computation, which is a key conceptual feature of our method. For example, assume the parties should store a large amount of data. Since all other methods require shares (of at least the same size as ours) being held by all parties, each party must store its shares somewhere, whereas in our case only the EVH and the KH need to store data. Thus, we improve on the amount of storage needed for three party protocols and match those of two party protocols.

The (mathematical) motivation to use helpers is that

the AND of two numbers can be computed by combining four parts consisting only of encrypted values and keys. In a naive computation, each part can be computed by one party, encrypted and combined by the KH and EVH to yield an encrypted AND of the two numbers. This would yield a total of four parties including two helpers. Through double encryption of values, we can reduce the number of parties to three. The derivation of the protocols uses the associative and distributive properties of linear secret sharing. Our scheme is illustrated for three encryption schemes based on XOR and addition. The given protocols perform efficient Boolean operations (AND, XOR) operations. They could also be extended to arithmetic operations (addition, multiplication). Linear secret sharing strikes through little computational overhead – in particular when compared to protocols that require cryptographic primitives, e.g. generation of prime numbers.

4 Encryption

We use linear secret sharing, but only two out of three parties obtain a share, i.e. in most situations we generate only two shares. We label these shares differently, i.e. we have one encrypted value and, potentially, multiple keys. The distinction between keys and encrypted values is sometimes helpful, e.g. for computing an exponential function (Section 7.2). For a given bit $m \in \{0,1\}$ we choose a random key $K \in \{0,1\}$. The encryption $ENC_K(m)$ of bit $m \in \{0,1\}$ using key K is the XOR (\oplus symbol), i.e. $ENC_K(m) := K \oplus m$. The decryption $DEC_K(c)$ of a ciphertext c is $DEC_K(c) := c \oplus K$.

5 XOR and NOT Operations

We discuss the entire process ranging from encryption of plaintexts to decryption of results for a single operation. To compute a (bitwise) XOR of two numbers a,b the client encrypts both a and b. It sends the two keys to the KH and the encrypted values to the EVH. As a next step, the KH computes the XOR of the two keys and the EVH the XOR of the two encrypted values. Both send their results back to the client. The client obtains $a \oplus b$ by decrypting the result from the EVH with the key received

from the KH.

A NOT operation (denoted by \neg) corresponds to computing an XOR of an expression and the constant one. It can be done by the EVH by computing the 'NOT' of the encrypted value. More mathematically, we have $\neg a = a \oplus 1$ and $\neg ENC_{K_a}(a) = \neg(a \oplus K_a) = (\neg a) \oplus K_a = ENC_{K_a}(\neg a)$.

6 Basic Ideas for an AND Gate

To illustrate the main ideas, we discuss the AND gate for two numbers using two helpers aside from the EVH and the KH. Later, we refine the algorithms to require only one helper, i.e. three parties, and extend this to multiple parties. Note, all our main results are based on the three party protocols using one helper. The protocol could be extended to multiplication working in an analogous manner. The key idea is to express the AND of the plaintexts using several parts, each consisting of an encrypted value and a key. Each part can be computed on a separate party, e.g. we use (and prove) that

$$a \wedge b = (ENC_{K_a}(a) \wedge ENC_{K_b}(b)) \oplus (K_a \wedge K_b)$$
$$\oplus (K_a \wedge ENC_{K_b}(b)) \oplus (K_b \wedge ENC_{K_a}(a)) \quad (1)$$

Each of the four terms $ENC_{K_a}(a) \wedge ENC_{K_b}(b)$, $K_a \wedge ENC_{K_b}(b)$, $K_b \wedge ENC_{K_a}(a)$ and $K_a \wedge K_b$ is computed by one party. Each of the two helpers chooses a key to encrypt its part before sharing the encrypted part with the EVH and the key with the KH. The KH and EVH combine all partial results to obtain the key and encrypted value of $a \wedge b$. The algorithm to compute the AND of two bits is shown in Figure 2. In Steps 1 to 3, the client prepares the computation. Step 4 can also be seen as part of the preparation, i.e. secret sharing. The actual computation of the AND consists only of steps 5-9. No keys must be transmitted during computation if keys are pre-shared or randomness is created using a shared secret seed and pseudo-random number generators (as we stated in the Model Section).

Next, we prove that the protocol is secure and correct. Security can also be shown - it follows since no party can reveal any information about a secret by any combination of the values it has obtained. We prove it formally only for our best protocols, e.g. for the three party protocol in Section 7.

Theorem 2. The AND protocol in Figure 2 is correct.

Proof. To show correctness, i.e. that we indeed compute $a \wedge b$, we must prove that the decryption done by the client yields the correct result. From Figure 2 we see that the final key delivered to the client is $K_f := t_3 \oplus K_3 \oplus K_4$. Thus, it remains to show that for the EVH holds the claimed equation in Step 9:

$$ENC_{K_f}(a \wedge b) = ENC_{t_3 \oplus K_3 \oplus K_4}(a \wedge b)$$

= $t_0 \oplus ENC_{K_2}(t_1) \oplus ENC_{K_4}(t_2)$ (2)

We prove Equation (1) first. It can be derived using basic laws such as distributiveness and associativeness, $x \oplus x = 0$ and $x \oplus 0 = x$ and, thus, $a \oplus x \oplus x = a$:

$$a \wedge b = (a \oplus K_a \oplus K_a) \wedge b$$

$$= ((a \oplus K_a) \wedge b) \oplus (K_a \wedge b)$$

$$= ((a \oplus K_a) \wedge (b \oplus K_b \oplus K_b)) \oplus (K_a \wedge (b \oplus K_b \oplus K_b))$$

$$= ((a \oplus K_a) \wedge (b \oplus K_b)) \oplus ((a \oplus K_a) \wedge K_b)$$

$$\oplus (K_a \wedge (b \oplus K_b)) \oplus (K_a \wedge K_b)$$
(3)

Note, by using the definition of the encryption $ENC_K(m) = m \oplus K$ we obtain Equation (1) from (3). Next, we prove the Equation (2) starting from $ENC_{K_f}(a \land b)$ substituting the final key $K_f := t_3 \oplus K_3 \oplus K_4$:

$$ENC_{K_f}(a \wedge b)$$

$$=ENC_{t_3 \oplus K_3 \oplus K_4}(a \wedge b) \text{ (Step 9, KH, Figure 2)}$$

$$=(a \wedge b) \oplus (t_3 \oplus K_3 \oplus K_4)$$

$$=(a \wedge b) \oplus (K_a \wedge K_b) \oplus K_3 \oplus K_4 \text{ (Using } t_3 := K_a \wedge K_b)$$

$$=((a \oplus K_a) \wedge (b \oplus K_b)) \oplus (K_a \wedge (b \oplus K_b))$$

$$\oplus (K_b \wedge (a \oplus K_a)) \oplus (K_a \wedge K_b) \oplus (K_a \wedge K_b)$$

$$\oplus K_3 \oplus K_4 \text{ (Using Eq. (3))}$$

$$=((a \oplus K_a) \wedge (b \oplus K_b)) \oplus (K_a \wedge (b \oplus K_b)) \oplus K_4$$

$$\oplus (K_b \wedge (a \oplus K_a)) \oplus K_3 \text{ (Using } a \oplus (x \oplus x) = a)$$

$$=(ENC_{K_a}(a) \wedge ENC_{K_b}(b)) \oplus ENC_{K_4}(K_a \wedge ENC_{K_b}(b))$$

$$\oplus ENC_{K_3}(ENC_{K_a}(a) \wedge K_b) \text{ (Using } ENC_K(m) := m \oplus K)$$

$$=t_0 \oplus ENC_{K_3}(t_1) \oplus ENC_{K_4}(t_2)$$

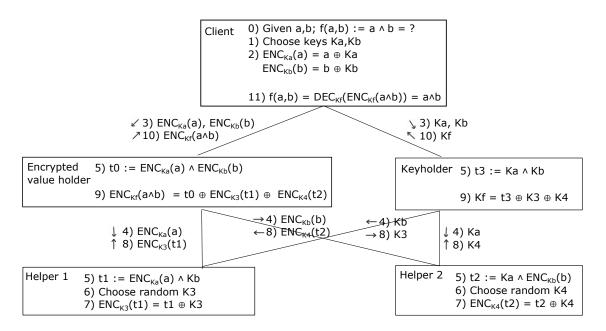


Figure 2: For comprehension purposes: Algorithm for an AND (\land) of two bits using two helpers. Later protocols use one helper only.

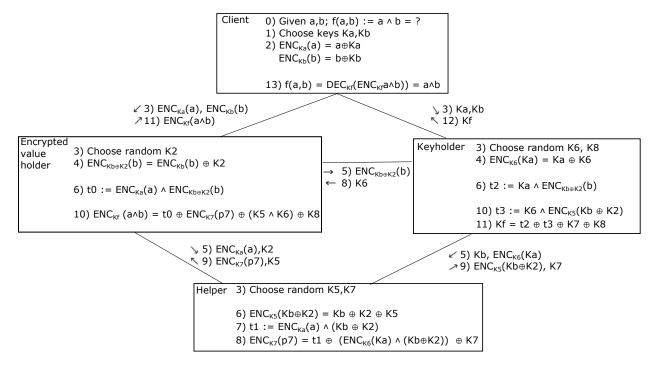


Figure 3: Algorithm for an AND (\land) operation of two bits using three parties.

7 Three Parties

Here, we reduce the number of parties from four to three but still focus on the computation of a single 'AND'. It is possible to use only one helper, i.e., three parties. We remove Helper 2. For an AND of two secrets a, b encrypted with K_a and K_b Helper 2 computes $ENC_{K_b}(b) \wedge K_a$ (see Figure 2). None of the other three parties can compute this expression using both $ENC_{K_b}(b)$ and K_a , since all parties hold at least one of the two values K_b , $ENC_{K_a}(a)$ and therefore they could reveal a secret. However, the remaining helper (i.e. Helper 1) can compute on encrypted values, i.e. the EVH can encrypt $ENC_{K_h}(b)$ with a randomly chosen key K_2 to obtain a 'double' encrypted value of b, i.e. $ENC_{K_b \oplus K_2}(b)$. The helper can use $ENC_{K_b \oplus K_2}(b)$ instead of $ENC_{K_h}(b)$. In particular, the EVH can double encrypt b and it can share $ENC_{K_b \oplus K_2}(b)$ with the KH (as long as the KH does not obtain K_2) and the key K_2 with the helper. This allows the KH to compute $K_a \wedge ENC_{K_b \oplus K_2}(b)$, leaving to compute $K_a \wedge (K_b \oplus K_2)$: We encrypt both values, i.e. K_a with K_6 and $K_b \oplus K_2$ with K_5 and compute using Equation (3):

$$K_{a} \wedge (K_{b} \oplus K_{2}) =$$

$$ENC_{K_{6}}(K_{a}) \wedge ENC_{K_{5}}(K_{b} \oplus K_{2}) \oplus K_{5} \wedge ENC_{K_{6}}(K_{a})$$

$$\oplus K_{6} \wedge ENC_{K_{5}}(K_{b} \oplus K_{2}) \oplus (K_{5} \wedge K_{6})$$

$$(4)$$

In this case, we do not need to distribute all four terms to (four) different parties. In our scenario the (remaining) helper holds $K_b \oplus K_2$, K_5 and $ENC_{K_6}(K_a)$. Thus, it can compute two terms, namely $ENC_{K_6}(K_a) \wedge ENC_{K_5}(K_b \oplus K_2)$ and $K_5 \wedge ENC_{K_6}(K_a)$. We can even simplify for the helper: $ENC_{K_6}(K_a) \wedge ENC_{K_5}(K_b \oplus K_2) \oplus K_5 \wedge ENC_{K_6}(K_a) = ENC_{K_6}(K_a) \wedge (K_b \oplus K_2)$.

This idea is realized in the protocol shown in Figure 3. The message complexity can be reduced by pre-sharing of keys. The very last key (K_8) is only needed if the result is used in further computations, e.g., we compute $(a \wedge b) \wedge c$ and reuse $(a \wedge b)$.

Next, we show correctness and security of the AND protocol.

Theorem 3. *The AND protocol in Figure 3 is correct.*

Proof. Analogously to the proof of Theorem 2, we show that decrypting the encrypted result (Step 10 for the EVH)

with the final key (Step 11 for KH) in Figure 3 gives $a \wedge b$. We start by transforming $a \wedge b$ as shown below:

```
a \wedge b = ((a \oplus K_a) \wedge (b \oplus (K_b \oplus K_2))) \oplus ((a \oplus K_a) \wedge (K_b \oplus K_2)) \oplus
            (K_a \wedge (b \oplus (K_b \oplus K_2))) \oplus (K_a \wedge (K_b \oplus K_2)) (Using Eq. 3)
         = ((a \oplus K_a) \land (b \oplus (K_b \oplus K_2))) \oplus ((a \oplus K_a) \land (K_b \oplus K_2))
            \oplus (K_a \wedge (b \oplus (K_b \oplus K_2)))
            \oplus ENC_{K_6}(K_a) \wedge ENC_{K_5}(K_b \oplus K_2) \oplus K_5 \wedge ENC_{K_6}(K_a)
            \oplus K_6 \wedge ENC_{K_5}(K_b \oplus K_2) \oplus (K_5 \wedge K_6) (Using Eq. 4)
        =ENC_{K_a}(a) \wedge ENC_{K_b \oplus K_2}(b) (Using ENC_K(m) = K \oplus m)
            \oplus ENC_{K_a}(a) \wedge (K_b \oplus K_2) \oplus K_a \wedge ENC_{K_b \oplus K_2}(b)
            \oplus ENC_{K_6}(K_a) \wedge ENC_{K_5}(K_b \oplus K_2)
            \oplus K_5 \wedge ENC_{K_6}(K_a) \oplus K_6 \wedge ENC_{K_5}(K_b \oplus K_2) \oplus (K_5 \wedge K_6)
        =t_0 (Using t_0 := ENC_{K_a}(a) \wedge ENC_{K_b \oplus K_2}(b)
             \oplus t_1 \ (Using \ t_1 := ENC_{K_a}(a) \land (K_b \oplus K_2))
            \oplus t_2 (Using \ t_2 := K_a \wedge ENC_{K_b \oplus K_2}(b))
            \oplus ENC_{K_6}(K_a) \wedge ENC_{K_5}(K_b \oplus K_2)
            \oplus K_5 \wedge ENC_{K_6}(K_a) \oplus K_6 \wedge ENC_{K_5}(K_b \oplus K_2) \oplus (K_5 \wedge K_6)
         =t_0 \oplus (K_5 \wedge K_6) \oplus t_1 \oplus K_7 \oplus K_7
            \oplus ENC_{K_6}(K_a) \wedge ENC_{K_5}(K_b \oplus K_2) \oplus K_5 \wedge ENC_{K_6}(K_a)
            \oplus t_2 \oplus K_6 \wedge ENC_{K_5}(K_b \oplus K_2)
             (Rearranging and XOR with K_7 \oplus K_7 = 0)
         =t_0 \oplus (K_5 \wedge K_6) \oplus t_1 \oplus K_7 \oplus K_7 \oplus ENC_{K_6}(K_a) \wedge (K_b \oplus K_2)
             \oplus t_2 \oplus t_3 (Simplifying and t_3 := K_6 \wedge ENC_{K_5}(K_b \oplus K_2))
        =t_0 \oplus (K_5 \wedge K_6) \oplus t_1 \oplus ENC_{K_6}(K_a) \wedge (K_b \oplus K_2) \oplus K_7
             \oplus t_2 \oplus t_3 \oplus K_7
        =t_0 \oplus (K_5 \wedge K_6) \oplus ENC_{K_7}(t_4) \oplus K_7 \oplus (t_2 \oplus t_3 \oplus K_7)
            \oplus K_8 \oplus K_8
             (Using Def. of ENC_{K_7}(t_4) in Step 8, Figure 3)
        =t_0 \oplus (K_5 \wedge K_6) \oplus ENC_{K_7}(t_4) \oplus K_7 \oplus K_8 \oplus K_f
             (Using K_f = t_2 \oplus t_3 \oplus K_7 \oplus K_8 in Step 10, Figure 3)
        =ENC_{K_f}(a \wedge b) \oplus K_f = a \wedge b (Using Step 10, Figure 3)
```

Theorem 4. The AND protocol in Figure 2 is perfectly secure.

Security follows since no party can reveal a secret by arbitrary combination of the values it has.

Proof. We show that none of the parties can obtain any information about *a* or *b*.

The EVH obtains keys K_2, K_3, K_5, K_6, K_8 and encrypted values $ENC_{K_a}(a), ENC_{K_b}(b), ENC_{K_7}(t_7)$. To get information about a or b the EVH needed to decrypt one of the encrypted values, i.e. it needed K_a or K_b or K_7 . However, it has no information about these keys. The KH obtains $K_a, K_b, K_6, K_7, K_8, ENC_{K_2 \oplus K_b}(b) = ENC_{K_2}(b \oplus K_b), ENC_{K_5}(K_b \oplus K_2)$. The KH has neither K_5 nor K_2 so it cannot disclose any information about a or b. The helper obtains K_b, K_2, K_5, K_7 and $ENC_{K_a}(a), ENC_{K_6}(K_a)$. Since the helper has no information about K_a and K_6 , it cannot learn anything about K_a or K_b

Theorem 5. The computation of the AND protocol (Steps 4-10) in Figure 2 needs a total of 5 transmitted bits.

Proof. In our model we assume that keys are pre-shared. The distribution of the secrets by the client is not part of the computation. Thus, only transmissions of encrypted values of Steps 5 to 9 are relevant, which yields a total of five bits, i.e. $ENC_{K_a}(a),ENC_{K_b \oplus K_2}(b)$, $ENC_{K_5}(K_a),ENC_{K_7}(t_7)$ and $ENC_{K_5}(K_b \oplus K_2)$.

7.1 Multiple ANDs: Reusing Variables and Multiple Encryptions

We have shown that a single AND operation is perfectly secure, underlying the assumption that no party has both an encrypted value and a matching key. However, when using the same variables in multiple operations (but in different pairings), the amortized communication per gate is reduced, since the KH and EVH only need to share some terms with the helper once. For example, to compute $a \wedge b$, $a \wedge c$ the values related to a need to be shared only once with the helper. In some cases, we might need two different encryptions. The need for two encryptions arises when evaluating circular structures, such as all three terms $a \wedge b$, $a \wedge c$ and $b \wedge c$. The encrypted values and keys cannot be distributed such that the helper gets an encrypted value without getting the corresponding key to decrypt it. More precisely, to compute $a \wedge b$ (see Figure 3), the helper gets the key K_b , the encrypted key $ENC_{K_6}(K_a)$ and the encrypted value $ENC_{K_a}(a)$. To compute $a \wedge c$ the helper must get K_c and the encrypted value $ENC_{K_c}(c)$, since it already received $ENC_{K_a}(a)$ and thus it cannot get key K_a . To obtain $b \wedge c$ is not possible, since the helper has already K_b and K_c and thus cannot get either $ENC_{K_c}(c)$ or $ENC_{K_b}(b)$.

To handle circular structures of the form above, two encryptions of the same variable suffice. Multiple encryptions of the same confidential variable can easily be created by the KH and EVH. To re-encrypt a variable a encrypted with key K_a . The KH chooses a key K'_a and transmits $ENC_{K'_a}(K_a)$ to the EVH, which computes $ENC_{ENC_{K'_a}}(K_a)$ ($ENC_{K_a}(a)$) = $ENC_{K'_a}(a)$. It is not hard to see that for each variable out of v variables we need to at most two encryptions to be able to compute any of the $O(v^2)$ possible pairs. For an AND involving variable a we use the first encryption of a, if a is on the left-hand side of the AND, e.g. for $a \wedge x$, and the second encryption, if it is on the right-hand side, e.g. $x \wedge a$.

Next, we discuss the adjusted protocols for the AND of two variables $a \wedge b$ using three parties reusing priorly shared values with the helper. There are three cases: Reusing both variables, reusing one variable, reusing one variable and reencrypting the other. The protocol in Figure 4 shows the reuse of both operands of an AND. In this case, compared to the protocol without reuse (Figure 3) only two keys K_7' and K_8' need to be generated and shared as well as the value $ENC_{K_7'}(t_4)$.

Reusing values for one variable a, while transmitting those of a variable b, works similarly. In the protocol in Figure 3 two bits, ie. $ENC_{K_a}(a)$ and $ENC_{K_6}(K_a)$, do not have to be transmitted. Reusing values for one variable a, while reencrypting the other is shown in Figure 5. In this case, we first reencrypt one variable, i.e. b, and then use the reencrypted values. The KH sends $ENC_{K_b'}(K_b)$ to the KH. The EVH reencrypts b to get $ENC_{K_b'}(b)$ and uses this value in his computations. Note, that the KH can reuse $ENC_{K_2 \oplus K_b}(b)$ to get $ENC_{K_2 \oplus K_b'}(b)$, i.e. the EVH and HE keep K_2 and the EVH does need to share $ENC_{K_b' \oplus K_2}(b)$ to the HE. The same holds for $ENC_{K_5}(K_b \oplus K_2)$.

Theorem 6. The protocols in Figure 4 and 5 are perfectly secure and correct.

Proof. Correctness follows from correctness of Figure 3, since all computations are identical. Security follows from the fact that no party obtains additional information. In the protocol of Figure 4 the helper obtains no input. The EVH obtains an encrypted value that was encrypted with a newly generated key K'_7 . The KH only obtains the newly generated key K'_7 . In the protocol of Figure 5 the

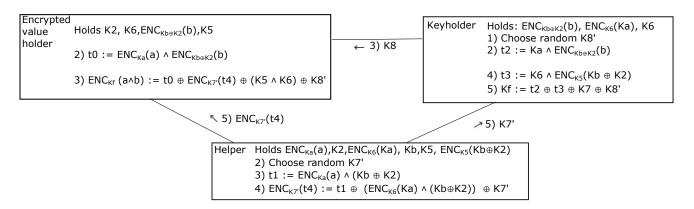


Figure 4: Algorithm for an AND (\wedge) operation of two bits a, b reusing priorly shared values with the helper.

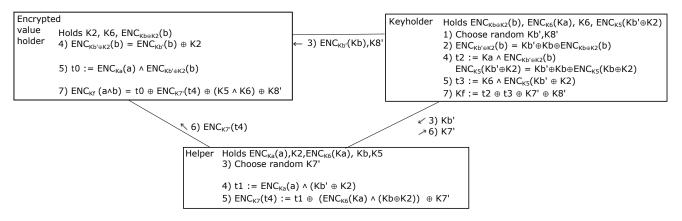


Figure 5: Algorithm for an AND (\land) operation of two bits a,b, reusing values for variable a and re-encrypting variable b.

EVH additionaly obtains the encrypted key $ENC_{K'_b}(K_b)$ and K'_8 but no information about K'_b . The helper obtains K'_b , but not $ENC_{K'_b}(K_b)$. The KH obtains K'_7 and K'_8 . \square

Theorem 7. For v variables computing $t(v) \in [v, v^2/2]$ pairs, needs at most $1 + 4 \cdot v/t(v)$ bits in total.

Proof. The protocol in Figure 3 needs 5 bits to be communicated to compute an AND of two bits due to Theorem 5. The protocol in Figure 5 needs 2 bits using pre-shared keys, i.e. $ENC_{K7'}(t_4)$ and $ENC_{K_b'}(K_b)$. The protocol in Figure 4 needs 1 bit, $ENC_{K7'}(t_4)$. We need at most two encryptions for each variable. Sharing all parts of an encryption needs two bits at most, yielding 4v for sharing. Thus, to compute t(v) terms, we need just one bit per term to return the result, giving a total of $4 \cdot v + t(v)$ bits.

7.2 Exponential Functions

We show how to compute the exponential function in the three party case ensuring statistical security. We assume that a secret a is encrypted using additive blending without modulo of a random key K, i.e. $ENC_K(a) = a + K$. Our protocol computes c^a for a public constant c and a confidential value a. The protocol relies on the well known identity $c^{a+b} = c^a \cdot c^b$. The EVH shares a random key K_1 with the KH. It computes $ENC_{K_1}(c^{a+K}) = c^{ENC_K(a)} + K_1$ and transmits this to the Helper. The KH computes $K_3 := -K_1/c^K - K_2$ and transmits K_3 to the Helper. The Helper receives K, K_3 , $ENC_{K_1}(c^{a+K})$. It chooses a key K_2 and shares the key with the KH. It computes $ENC_{K_1}(c^{a+K})/c^K - K_3 = c^a + K_2 = ENC_{K_2}(e^a)$, which is shared with the EVH.

7.3 Arbitrary Fan-in

We can compute $a_0 \wedge a_1 \wedge ... \wedge a_{w-1}$ using two rounds only. Similarly as for two variables, the AND of multiple variables can be expressed using multiple terms such that each term consists of ANDs of multiple encrypted values and multiple keys. The EVH can compute the AND of all encrypted values locally, and the KH can do the same for the keys. Thereby, reducing each term to one partial result held by the KH and one by EVH. These two partial results can then be ANDed using the prior protocol for

two variables, yielding the AND of one term. The results of all terms are XORed.

Theorem 8. A gate $a_0 \wedge a_1 \wedge ... \wedge a_{w-1}$ can be evaluated in 2 rounds using messages of size $O(2^w)$ for an arbitrary parameter $k \in [2, w]$ and $O(w2^w)$ bit operations.

Proof. We can express the AND using 2^w terms by generalizing Equation (1). Let S_w be all subsets of $\{0, 1, ..., w-1\}$. We have

$$a_{0} \wedge a_{1} \wedge \ldots \wedge a_{w-1}$$

$$= (a_{0} \oplus K_{0} \oplus K_{0}) \wedge a_{1} \wedge \ldots \wedge a_{w-1}$$

$$= (ENC_{K_{0}}(a_{0}) \wedge a_{1} \wedge \ldots \wedge a_{w-1}) \oplus (K_{0} \wedge a_{1} \wedge \ldots \wedge a_{w-1})$$

$$= (ENC_{K_{0}}(a_{0}) \wedge (a_{1} \oplus K_{1} \oplus K_{1}) \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge a_{1} \wedge \ldots \wedge a_{w-1})$$

$$= (ENC_{K_{0}}(a_{0}) \wedge ENC_{K_{1}}(a_{1}) \wedge \ldots \wedge a_{w-1})$$

$$\oplus (ENC_{K_{0}}(a_{0}) \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge a_{1} \wedge \ldots \wedge a_{w-1})$$

$$= (ENC_{K_{0}}(a_{0}) \wedge (ENC_{K_{1}}(a_{1})) \wedge \ldots \wedge a_{w-1})$$

$$\oplus (ENC_{K_{0}}(a_{0}) \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge (a_{1} \oplus K_{1} \oplus K_{1}) \wedge \ldots \wedge a_{w-1})$$

$$\oplus (ENC_{K_{0}}(a_{0}) \wedge ENC_{K_{1}}(a_{1}) \wedge \ldots \wedge a_{w-1})$$

$$\oplus (ENC_{K_{0}}(a_{0}) \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

$$\oplus (ENC_{K_{0}}(a_{0}) \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge ENC_{K_{1}}(a_{1}) \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

$$\oplus (K_{0} \wedge K_{1} \wedge \ldots \wedge a_{w-1})$$

In the last step we applied to all a_i the same transformation as for a_0 and a_1 , i.e. replacing a_i by $a_i \oplus K_i \oplus K_i$ followed by an expansion of terms. We rearranged using the commutative property of the AND operation. It can easily seen that each of the w variables doubles the number of terms, yielding 2^w terms, i.e. each corresponding to one of the subsets S_w .

In Equation (5) each term t_i consists of ANDed values. It can be partitioned into two parts, one consisting of encrypted values t_E and one of keys t_K , e.g. for $t = ENC_{K_0}(a_0) \wedge ENC_{K_1}(a_1) \wedge K_2$ we get $t_E = ENC_{K_0}(a_0) \wedge ENC_{K_1}(a_1)$ and $t_K = K_2$. The EVH can compute the term t_E by computing the AND of all encrypted values without communication and the KH the term t_K in the same

manner. The EVH encrypts the locally computed term t_E and the KH encrypts t_K , i.e. the EVH chooses key K_{tE} , computes $ENC_{K_{tE}}(t_E)$ and sends the key K_{tE} to KH. The KH chooses K_{tK} , computes $ENC_{K_{tK}}(t_K)$ and sends the encrypted value $ENC_{K_{tK}}(t_K)$ to the EVH. Then they run the protocol (Figure 3) to AND the two terms $t_E \wedge t_K$. They do this for all 2^w terms in parallel. Finally, the EVH computes the XOR of all encrypted results for all 2^w terms and the KH computes the XOR of all keys, which yields the final result for each party.

8 Related Work

Helpers are not uncommon in MPC, e.g. [11, 16]. But they are often used as trusted entities. In this work, we do not trust the helper more than any other party. In the setting of [16] a client wants to know if a value held by the party matches her secret string. A helper assists in answering the query. The result of the query should also remain secret to the party. Generation of RSA keys is discussed in [11] using a helper. The helper is used to compute the product of primes using an interpolation of a quadratic polynomial. We integrate a helper on a much lower level of computation and adjust basic protocols like AND and XOR to use a helper.

Though a large body of work [20, 5, 1, 4, 8, 9] does not distinguish between encrypted values and keys, the idea of drawing such a separation has been employed in other contexts, eg. in the work of [10] discussing MPC the idea of using such a separation with public keys for voting schemes is mentioned.

Three parties are commonly used, e.g. [26, 9, 23]. The work [26] builds upon garbled circuits, essentially showing that garbled circuits can be made robust against corruption of one party. Sharemind [9] uses three parties and additive secret sharing, i.e. for a secret x each party P_i obtains a share x_i such that $\sum x_i \mod 2^{32} = x$. To perform a multiplication they compute all 6 shares $x_i \cdot x_j$ using [16]. A multiplication requires 3 rounds and 27 messages each containing a 32-bit value. The paper [9] also discusses why Shamir's secret sharing fails on the ring of 2^{32} (and needs more messages on the ring Z_p). The work [16] also uses an untrusted third party, which assists in the computation of approximate distances (e.g. of strings) using various metrics. The system of [23] uses

three parties and linear secret sharing to compute all nine shares for evaluation of multiplication as [9]. Recently, a new three party protocol[1] was introduced. In contrast to this work (and similar to other works, e.g., [9, 25]), each party obtains a share using linear secret sharing. The protocol creates correlated randomness among all three parties. We use correlated randomness for pairs of parties. They do not require a round for secret sharing as we do, but assuming that each variable appears on average (somewhat) more than 10 times in a circuit, their protocol [1] requires more communication to evaluate the circuit. In particular, if the number of terms t(v) is more than linear as the number of variables, we need only 2+o(1) bits per gate and thus outperform by a factor of 3.

An unbounded fan-in AND gate can be simulated [4] in (expected) constant number of rounds for arithmetic gates. They encrypt a number a_i held by party i as $ENC(a_i) = R_i \cdot a_i \cdot R_{i-1}^{-1}$ with R_i being a matrix of random elements. The product of all terms a_i is one element in the matrix being the product of all encryptions. To generate matrices of sufficient rank, they generate more than n^2 random matrices. We do not use multiplicative inverses in a group. We follow a different approach based on term expansions. Bar-Ilan et al. [4] requires messages that are of size proportional to the size of a constant depth, unbounded fan-in circuit for the function to evaluate. Our scheme (JOS) requires asymptotically also a constant number of rounds for computation of a w fan-in gate but more communication for large w. For the three party case, JOS outperforms [4] for small fan-in gates. For example, for w = 4 Bar-Ilan et al. requires at least 6 rounds (using k = 3), whereas JOS needs at most five. The total amount of communication of [4] is at least $129 \cdot l$ in contrast to $60 \cdot l$ of our scheme. Furthermore, it needs more local computation.

The BenOr-Goldwasser-Wigderson (BGW) [8, 2] gives several fundamental MPC protocols. Genaro-Rabin-Rabin (GRR) [18] simplifies BGW. GRR requires $n \ge 2 \cdot t + 1 \ge 3$ parties tolerating collusion of t parties, BGW can handle collusion of t < n/3 parties in the distrust all model. GRR and BGW use Shamir's secret sharing to derive a protocol for multiplication. The multiplication protocol Simple-Mult in GRR takes two secrets α and β shared by two polynomials $f_{\alpha}(x)$ and $f_{\beta}(x)$ to compute $\alpha \cdot \beta$. Party i computes the value $f_{\alpha}(i) \cdot f_{\beta}(i)$ using a ran-

dom polynomial. Then each party aggregates the input of other parties and reduces the size of the polynomial through interpolation to compute his share of $\alpha \cdot \beta$. A protocol for multiplication and addition using similar ideas as GRR but using additive secret sharing (without modulo) is given in [25]. In the case of three parties a secret a is split into three parts a_0, a_1, a_2 such that the sum equals a. In [25] each party gets two distinct parts. Multiplication of two secrets a and b is analogous to GRR by computing all nine pairs $a_i \cdot b_i$, aggregating them locally and sharing the result using independent randomness. A party then aggregates all received numbers to obtain the result $a \cdot b$. To compute $(a \cdot b) \cdot c$ each party would send its share of $a \cdot b$ to one other party, such that each party again holds two shares of the result. A key disadvantage of [25] is that shares double in size after every multiplication, making it impractical for even a modest number of multiplications. The paper by Yao [29] from the late 80ies still forms the underpinning for many works evaluating Boolean circuits. Yao showed how one party A can evaluate a private boolean circuit with private inputs from itself and another party B such that A does not learn anything about the inputs of B and B does not learn anything about the circuit or the input of A. To do so A computes a so called "garbled circuit" which is an encryption of the circuit containing the input. Afterwards, party B evaluates the encrypted circuit using its input and returns the result. Encryption encompasses encrypting every entry of the truth table of the boolean circuit and uses several algorithmic ideas such as oblivious transfer of keys to do the two party computation. The original scheme [29] allowed for a circuit only to be evaluated once without revealing information about the circuit. Since then a lot of improvements have been made, e.g., [17, 19, 7]. Reusable circuits come only with additive overhead in the form of a polynomial in the security parameter and circuit depth [19]. Our advantage compared to [19] is that we ensure perfect security and encryption is much simpler (and faster). Additionally, our communication complexity does not depend on a polynomial depending on the security parameter as well as the circuit depth, which can easily dominate the communication costs. Yao's scheme has been generalized to multiple parties by computing a common garbled circuit in BMR [5]. Several evaluations are possible using multilinear jigsaw puzzles [17]. A circuit can be garbled such that its encryption occurs only additional overhead [19]. Recent implemenations[7] allow for a single AES call per garbled-gate (justified in the random-permutation model). Goldreich-Micali-Widgerson (GMW) [20] uses oblivious transfer to compute any Boolean circuit. Values are encrypted such that each party holds parts of the non-encrypted value. The GMW protocol has round complexity linear in the depth of the circuit. Oblivious transfer has been continuously optimized, e.g. [3] uses symmetric cryptography. Still, using [3] for an oblivious transfer requires (as a lower bound) at least the size of the security parameter, which is significantly more than our total communication for an *AND*.

A significant body of work has focused on optimizing either the computational or communication overhead (e.g. [13, 15, 14, 21]) of MPC focusing on entire circuits for various security models using known schemes for evaluating gates. We focus on optimizing elementary operations for a single gate for perfect security that can be used to compute entire circuits. There is a vast number of secret sharing schemes, e.g. for a survey see [6]. Our linear encryption schemes are known. For instance, [22] encrypts a secret using XOR. Additive encryption as done in JOS roughly corresponds to [8] and has been also employed by [12]. Whereas prior work shared a secret with all parties, we use a dedicated helper to support computation and use the properties of the encryption schemes to derive novel protocols.

The first work attempting to compute exponential functions in a constant number of rounds was [24]. The model of [24] assumes that each party has a secret. Although they claim that an adversary can corrupt two parties for all their protocols, in fact, information about a secret (of one party) is revealed if one party behaves dishonestly. To see this consider, e.g. the protocol 3.1 for multiplication in [24]. They compute $u_1 + u_2 = x_1 \cdot x_2$ using a nonspecified algorithm, where x_1, x_2 are secrets, u_i are secret shares and u_i, x_i are held by party i. If an attacker corrupts parties 1 and 2 and obtains u_1 and u_2 it also obtains $x_1 \cdot x_2$. Clearly, revealing $x_1 \cdot x_2$ is a violation of confidentiality for both x_1 and x_2 . Furthermore, they require a two-party protocol (as blackbox) for real numbers to get u_1, u_2 such that $u_1 + u_2 = x_1 \cdot x_2$ but do not state any efficient protocol. This work is an extension of [27] with more detailed model discusion, proofs, computation of the exponential function, a protocol for reusing values and a protocol for achieving a trade-off between round and communication complexity for large fan-in gates. Prior work [28] based on [27] has shown how to compute various logical and numerical functions, e.g. trigonometric functions and divisions, as well as transformations between different encryption schemes.

9 Conclusions

We have assessed the idea of using one party as a helper in the context of secure-multi party computation. The derived protocols achieve little communication, storage, and computational overhead. In some cases, they are theoretically optimal regarding communication complexity, showing that a helper can be of great value from a theoretical perspective. Numerical computations relying on statistical security can in some cases also largely benefit from a helper.

References

- [1] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 805–817. ACM, 2016.
- [2] G. Asharov and Y. Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, pages 1–94, 2011.
- [3] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the* 2013 ACM SIGSAC conference on Computer & communications security, pages 535–548. ACM, 2013.
- [4] J. Bar-Ilan and D. Beaver. Non-cryptographic faulttolerant computing in constant number of rounds of interaction. In *Proceedings of the eighth annual* ACM Symposium on Principles of distributed computing, 1989.
- [5] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of*

- the twenty-second annual ACM symposium on Theory of computing, pages 503–513, 1990.
- [6] A. Beimel. Secret-sharing schemes: a survey. In *Coding and cryptology*, pages 11–46. Springer, 2011.
- [7] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key block-cipher. In *Security and Privacy (SP)*, 2013 IEEE Symposium on, pages 478–492. IEEE, 2013.
- [8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic faulttolerant distributed computation. In *Proceedings of* the twentieth annual ACM symposium on Theory of computing, pages 1–10, 1988.
- [9] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security-ESORICS 2008*, pages 192–206. Springer, 2008.
- [10] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. In *Finan-cial Cryptography*, volume 5628, pages 325–343. Springer, 2009.
- [11] D. Boneh and M. Franklin. Efficient generation of shared rsa keys. In *Advances in Cryptology* (*CRYPTO*), pages 425–439. 1997.
- [12] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks*. 2010.
- [13] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology–EUROCRYPT 2010*, pages 445–465. Springer, 2010.
- [14] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology-CRYPTO 2007*, pages 572–590. Springer, 2007.

- [15] I. Damgård and S. Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography*, pages 621–641. Springer, 2013.
- [16] W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *E-Commerce Security and Privacy*, pages 87–111. Springer, 2001.
- [17] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS)*, pages 40–49, 2013.
- [18] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc.* of the 17th ACM symposium on Principles of distributed computing, pages 101–111, 1998.
- [19] C. Gentry, S. Gorbunov, S. Halevi, V. Vaikuntanathan, and D. Vinayagamurthy. How to compress (reusable) garbled circuits. *IACR Cryptology ePrint Archive*, 2013:687, 2013.
- [20] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of 19th Symp. on Theory of computing*, pages 218–229, 1987.
- [21] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography*, pages 294–314. Springer, 2009.
- [22] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. *Electron*-

- ics and Communications in Japan (Part III: Fundamental Electronic Science), 72(9):56–64, 1989.
- [23] J. Launchbury, D. Archer, T. DuBuisson, and E. Mertens. Application-scale secure multiparty computation. In *Programming Languages and Sys*tems, pages 8–26. Springer, 2014.
- [24] W. Luo and X. Li. A study of secure multi-party elementary function computation protocols. In *Proceedings of the 3rd international conference on information security*, pages 5–12. ACM, 2004.
- [25] U. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.
- [26] P. Mohassel, M. Rosulek, and Y. Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proc. of the 22nd ACM Conf. on Computer and Communications Security*, pages 591–602, 2015.
- [27] J. Schneider. Lean and fast secure multi-party computation: Minimizing communication and local computation using a helper. *13th Int. Conf. on Security and Cryptography(SECRYPT)*, 2016.
- [28] J. Schneider and B. Lu. Secure numerical and logical multi party operations. *Journal of Information Security and Applications*, 2017.
- [29] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science(FOCS)*, 1986.