Lean and Fast Secure Multi-Party Computation: Minimizing Communication and Local Computation Using A Helper[1]

Johannes Schneider
ABB Corporate Research
Baden-Daettwil, Switzerland

Abstract

A client wishes to outsource computation on confidential data to a network of servers. He does not trust a *single* server, but believes that multiple servers do not collude. To solve this problem we introduce a new scheme called *JOS* for perfect security in the semi-honest model that naturally requires at least three parties. It differs from classical secure multi-party computation (MPC) through three points: (i) a client-server setting, where all inputs and outputs are only known to the client; (ii) the use of three parties, where one party serves merely as "helper" for computation, but does not store any shares of a secret; (iii) distinct use of the distributive and associative nature of well-known linear encryption schemes to derive our protocols. We improve on the total amount of communication needed to compute both an *AND* and a multiplication compared to all prior schemes (even two party protocols), while matching round complexity or requiring only one more round. For big-data analysis, network bandwidth is often the most severe limitation, thus minimizing the amount of communication is essential. Therefore, we make an important step towards making MPC more practical. We also reduce the total amount of storage needed (eg. in a database setting) compared to all prior schemes using three parties. Our local computation requirements lag behind non-encrypted computation by less than an order of magnitude per party, while improving on other schemes, ie. GRR, by several orders of magnitude. We also discuss a trade-off for round complexity and communication for large fan-in gates. We can compute an (unbounded) fan-in gate of w variables in constant time, ie. it can be evaluated in $O(\log k)$ rounds using messages of size $O(2^{w-k})$ for an arbitrary parameter $k \in [2, w]$ and $O(|C| \cdot 2^{w-k})$ bit operations. We also provide an extension to more than three parties.

Keywords: big data, client-server computation, secure cloud computing, secure multi-party computation, privacy preserving data mining

[1]This is an extended version of a conference paper[27].

1. Introduction

Cloud computing has boosted the need for methods to compute on encrypted data. Secure multi-party computation (MPC) is a computationally efficient approach that allows a client to outsource computation to a group of servers, relying on the assumption that servers do not collude (to retrieve confidential information). In the semi-honest model, a curious but passive attacker can monitor a server completely, eg. its memory, disc or CPU registers. Also these corrupted servers stick to the proposed protocol.
Our effort to minimize communication and local computation is essential to improve acceptance of MPC in industry. All modern schemes for data analysis, such as Hadoop or Spark, rely on data-parallelism, ie. 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). For this essential scenario of privacy preserving data mining, the bottleneck becomes bandwidth (ie. the amount of information that can be transmitted) and not network latency (ie. number of communication rounds). We address this concern by minimizing the number of exchanged bits in our protocols. Thus, we believe that our work is an important step towards making MPC more practical.

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. The keyholder stores keys but it has no access to ciphertexts. The encrypted value holder stores encrypted values but no keys. A helper assists computations. It might obtain keys and encrypted values that do not match, ie. none of its keys can be used to decrypt any of its encrypted values. The helper does not hold any shares per se, which is a key conceptual feature of our method. For example, assume the servers 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. Each part can be computed by one party, re-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. 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. We were able to design protocols that perform efficient Boolean operations (AND, XOR) and arithmetic operations (addition, multiplication). Linear secret sharing also strikes through little computational overhead – in particular, when compared to protocols that require cryptographic primitives, eg. we do not need to generate prime numbers, exponentiations of large numbers etc. Secure operations only need aside from random numbers a few additions and multiplications of numbers, which are only of the same length as the plaintext.

In summary, we make the following contributions:

- We present a new scheme for MPC in the semi-honest model for Boolean circuits as well as for arithmetic expressions using $n \geq 3$ parties. To the best of our knowledge, we improve on the amount of communication needed to compute an *AND* or to conduct a multiplication compared to all prior schemes (see Table 2). The round complexity of JOS matches other schemes or lags behind by at most one round more. We require less storage than all prior three party protocols, while matching storage requirements for two party protocols.

- Our protocols are efficient in terms of local computation. A secure addition requires adding shares only. A secure multiplication requires in total 6 multiplications of numbers of the same size as well as a few additions and random keys (analogously for AND, XOR). In particular, our evaluation shows that local computation overhead is low. We lag behind computation on non-encrypted data by a factor less than 10. Compared to GRR we achieve improvements in run-time measured in terms of local computation of 3 to 4 orders of magnitude for encryption and decryption and a factor of more than 20 for addition and multiplication.

- 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 $O(w \cdot 2^{w-k})$ bit operations. Results for multiplication are analogous.

- We give an extension to $n > 3$ parties.

2. Related Work

Helpers as trusted entities are not uncommon in MPC, eg. [9, 14]. In the setting of [14] a client wants to know if a value held by the server matches her secret string. A "helper" assists in answering the query. The result of the query should also remain secret to the server. Generation of RSA keys is discussed in [9] 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.

Three parties are commonly used, eg. [25, 8, 21]. The work [25] builds upon garbled circuits, essentially showing that garbled circuits can be made robust against corruption of one party. Sharemind [8] uses three parties and additive secret sharing, ie. 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 [14]. A multiplication requires 3 rounds and 27 messages each containing a 32-bit value. We require at most 2 rounds and 4 messages. The paper [8] 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 [14] uses also an untrusted third party, which assists in the computation of approximate distances (eg. of strings) using various metrics. The system of [21] uses three parties and linear secret sharing to compute all nine shares for evaluation of multiplication as [8].

An unbounded fan-in AND gate can be simulated [3] 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, but we follow a different approach based on term expansions. Bar-Ilan et al [3] 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 for more communication large w. For the three party case, JOS outperforms [3] for small fan-in gates. For example, for $w = 4$ Bar-Ilan et al requires at least 6 rounds, whereas JOS needs at most five (using $k = 3$). The total amount of communication of [3] 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) [7, 1] gives several fundamental MPC protocols. Genaro-Rabin-Rabin (GRR) [16] simplifies BGW. GRR requires $n \geq 2 \cdot t + 1 \geq 3$ parties tolerating collusion of t parties, BGW can handle collusion of $t < n/3$ parties. GRR and BGW use Shamir's secret sharing to derive a protocol for multiplication. For a performance-based comparison showing the practical gains of JOS, consult Tables 1 and 2. Scaling up to n parties multiplying k bit numbers GRR needs $O(n^2 k \log n)$ or $O(nk^2)$ bit operations [22]. This matches our complexity asymptotically. 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 random 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 [24]. 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 [24] each party gets two distinct parts. Multiplication of two secrets a and b is analogous to GRR by computing all nine pairs $a_i \cdot b_j$, 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 [24] is that shares double in size after every multiplication, making it impractical for even a modest number of multiplications. This is in contrast to GRR which reduces the size of shares (using computationally costly interpolation) and to our scheme, which also does not grow the size of shares.

In contrast, we encrypt a using a single randomly chosen key, yielding two shares $a + k$ and k. This leaves one party (the helper) without a share, which is not the case for any other discussed protocols.

The paper by Yao [28] 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 its own 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 [28] allowed for a circuit only to be evaluated once without revealing information about the circuit. A lot of improvements have been made of several aspects of the protocol, eg. [15, 17, 6]. Reusable circuits come only with additive overhead in the form of a polynomial in the security parameter and circuit depth [17]. Our advantage compared to [17] 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 [4]. Several evaluations are possible using multilinear jigsaw puzzles [15]. A circuit can be garbled such that its encryption occurs only additional overhead [17]. Recent implemenations[6] allow for a single AES call per garbled-gate (justified in the random-permutation model). Goldreich-Micali-Widgerson (GMW) [18] 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, eg. [2] uses symmetric cryptography. Still, using [2] 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 (eg. [11, 13, 12, 19]) 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, eg. for a survey see [5]. Our linear encryption schemes are known. For instance, [20] encrypts a secret using XOR. Additive encryption as done in JOS roughly corresponds to [7] and has been also employed by [10]. 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.

3. Model And Network Architecture

A client holds an arbitrary amount of secret values (potentially, more than the number of parties n) and wishes to evaluate a function using n servers, ie. parties, such that no party learns anything about the input or the output.[2] Thus, typically, a client encrypts its secrets and distributes the shares among the servers, the servers compute the function and return their shares to the client. The client itself does not participate in the computation and, therefore, does not count as party. This differs from the classical MPC model, where each party holds a secret (or at least a share) and the output should

[2]Aside from unconditional security, we also discuss purely additive encryption(without modulo) that yields statistical security only.

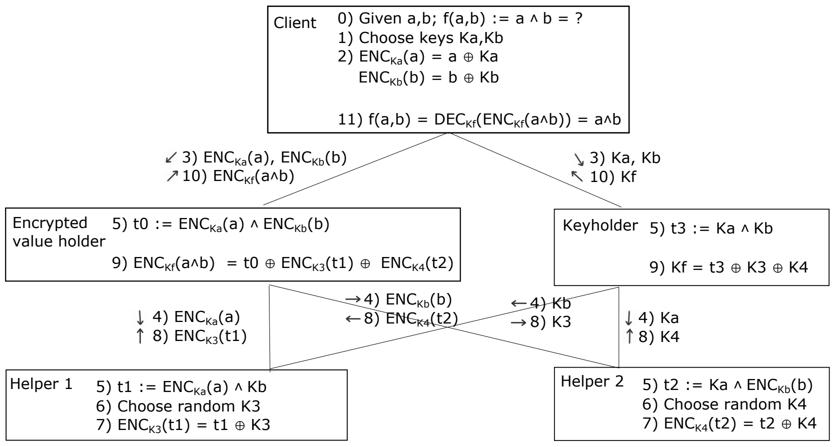


Figure 1: Algorithm for an AND ($\wedge$) of two bits using four parties.

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) and, 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.

We consider the semi-honest model with $n = 3$ parties out of which at most $n - 2$ parties can be corrupted by an adaptive adversary.[3] We consider the semi-honest model with $n = 3$ parties out of which at most $n - 2$ parties can be corrupted by an adaptive adversary. 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. Generally, the KH holds keys and the EVH holds encrypted values. Note, there are circumstances, where this distinction has to be seen less strict, eg. for secret a there might be two encryptions such that one encrypted value is held by the KH. A network with three parties is shown in Figure 2. We assume perfectly secure communication channels between parties.

4. Encryption

We use linear secret sharing, but only two out of three parties obtain a share, ie. we generate only two shares. We label these two shares differently, ie. key and encrypted value.[4]

Encryption for Boolean Operations: 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

[3]In an extended version, we discuss the case $n \geq 3$.

[4]The distinction between keys and encrypted values is sometimes helpful, eg. for computing the sine function [26].

($\oplus$ symbol), ie. $ENC_K(m) := K \oplus m$. The decryption $DEC_K(c)$ of a ciphertext c is $DEC_K(c) := c \oplus K$.
Encryption for Arithmetic Operations: For a given number $m \in \{0,1\}^l$ of l bits we choose a random key $K \in \{0,1\}^b$ using $b \geq l$ bits. The encryption $ENC_K(m)$ of number m using key K is then $ENC_K(m) := (K+m) \mod 2^b$. We denote inverse elements with superscripts, eg. the inverse of K is $K^{-1} := 2^b - K$. The decryption $DEC_K(c)$ of a ciphertext c is $DEC_K(c) := (c + K^{-1}) \mod 2^b$. Whereas the above encryption in an arithmetic ring guarantees unconditional security, we also mention a variant without modulo that is only statistically secure. If we just add the key, we have: $ENC_K(m) := K+m$, $DEC_K(ENC_K(m)) = (m+K)+K^{-1} = m+K-K = m$.

5. Addition, Additive Inverse, XOR and NOT

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 $\neg ENC_{K_a}(a) = \neg(a \oplus K_a) = (\neg a) \oplus K_a = ENC_{K_a}(\neg a)$.

Additions of two (or more) numbers works analogously to the XOR operation. To deal with negative numbers we must use the two complement, ie. a number $a \in [-2^{b-1}, 2^{b-1}-1]$ is represented as $(2^{b-1}+a) \mod 2^b$. To compute the additive inverse the EVH can set the sign bit $b-1$ by adding 2^{b-1}, ie. $ENC_{K_a}(2^{b-1}+a) = (2^{b-1} + ENC_{K_a}(a)) \mod 2^b$.

6. Basic Ideas For Multiplication and AND

To illustrate the main ideas, we discuss the AND and multiplication of two numbers using two helpers aside from the EVH and the KH. Later, we refine the algorithms to require only one helper, ie. three parties. Note, all our main results are based on the three party protocols using one helper. Multiplication and AND work 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, eg. we use (and prove) that

$$\begin{aligned} a \wedge b = &\big(ENC_{K_a}(a) \wedge ENC_{K_b}(b)\big) \oplus (K_a \wedge K_b) \\ &\oplus \big(K_a \wedge ENC_{K_b}(b)\big) \oplus \big(K_b \wedge ENC_{K_a}(a)\big) \end{aligned} \tag{1}$$

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 1. As we focus on a client-server setting, we minimize the effort for the client and let the parties do secret sharing among themselves. In traditional MPC, this task would be performed by the client as part of Step 3. For example, in Figure 1 Step 4 is part of the secret sharing and could also be performed by the client. Thus, the actual computation of the multiplication consists only of steps 5-9. No keys must be transmitted during computation if keys are pre-shared.

Theorem 1. *The AND protocol in Figure 1 is perfectly secure and the result is correct.*

Proof. Since no party receives both an encrypted value and the uniquely corresponding key the protocol is perfectly secure.

To show correctness, ie. that we indeed compute $a \wedge b$, we must prove that the decryption done by the client yields the correct result. From Figure 1 we see that the final key delivered to the client is $Kf = t3 \oplus K3 \oplus K4$. Thus, it remains to show that for the EVH holds the claimed equation in Step 9:

$$\begin{aligned} ENC_{Kf}(a \wedge b) &= ENC_{t3 \oplus K3 \oplus K4}(a \wedge b) \\ &= t0 \oplus ENC_{K3}(t1) \oplus ENC_{K4}(t2) \end{aligned} \tag{2}$$

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

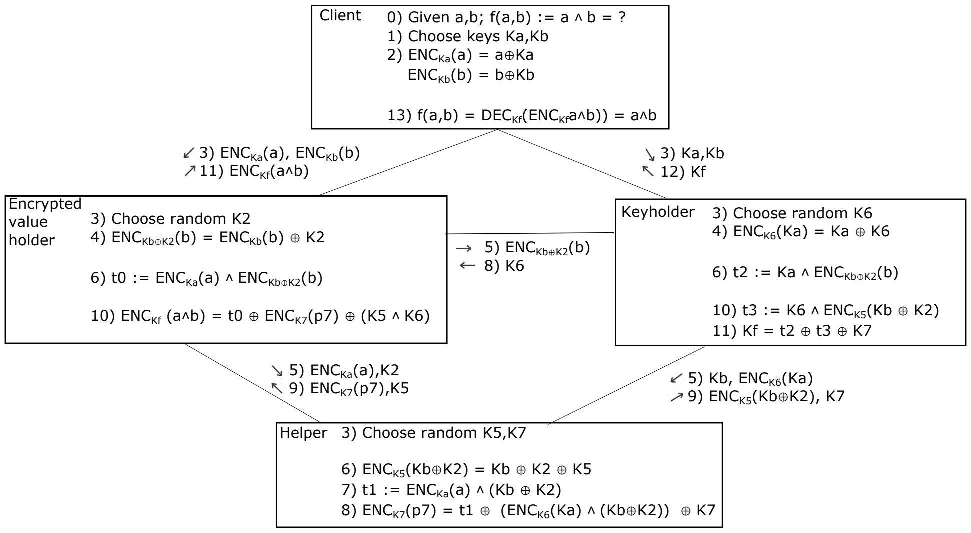
$$\begin{aligned} a \wedge b =& (a \oplus K_a \oplus K_a) \wedge b \\ =& \big((a \oplus K_a) \wedge b\big) \oplus \big(K_a \wedge b\big) \\ =& \big((a \oplus K_a) \wedge (b \oplus K_b \oplus K_b)\big) \oplus \big(K_a \wedge (b \oplus K_b \oplus K_b)\big) \\ =& \big((a \oplus K_a) \wedge (b \oplus K_b)\big) \oplus \big((a \oplus K_a) \wedge K_b\big) \\ & \oplus \big(K_a \wedge (b \oplus K_b)\big) \oplus \big(K_a \wedge K_b\big) \end{aligned} \tag{3}$$

Note, by using the definition of the encryption $ENC_K(m) = m \oplus K$ we obtain Equation (1) from (3). Starting from $ENC_{Kf}(a \wedge b)$ substituting the final key $Kf = t3 \oplus K3 \oplus K4$

we prove the initial Equation (2):

$$
\begin{aligned}
& ENC_{Kf}(a \wedge b) \\
=& ENC_{t3 \oplus K3 \oplus K4}(a \wedge b) \text{ \textit{(Step 9, KH, Figure 1)}} \\
=& (a \wedge b) \oplus (t3 \oplus K3 \oplus K4) \\
=& (a \wedge b) \oplus (K_a \wedge K_b) \oplus K3 \oplus K4 \text{ \textit{(Using} } t3 := K_a \wedge K_b\text{\textit{)}} \\
=& \big((a \oplus K_a) \wedge (b \oplus K_b)\big) \oplus \big(K_a \wedge (b \oplus K_b)\big) \\
& \oplus \big(K_b \wedge (a \oplus K_a)\big) \oplus (K_a \wedge K_b) \oplus (K_a \wedge K_b) \\
& \oplus K3 \oplus K4 \text{ \textit{(Using Eq. (3))}} \\
=& \big((a \oplus K_a) \wedge (b \oplus K_b)\big) \oplus \big(K_a \wedge (b \oplus K_b)\big) \oplus K4 \\
& \oplus \big(K_b \wedge (a \oplus K_a)\big) \oplus K3 \text{ \textit{(Using} } a \oplus (x \oplus x) = a\text{\textit{)}} \\
=& \big(ENC_{K_a}(a) \wedge ENC_{K_b}(b)\big) \oplus ENC_{K4}(K_a \wedge ENC_{K_b}(b)) \\
& \oplus ENC_{K3}(ENC_{K_a}(a) \wedge K_b) \text{ \textit{(Using} } ENC_K(m) = m \oplus K\text{\textit{)}} \\
=& t0 \oplus ENC_{K3}(t1) \oplus ENC_{K4}(t2)
\end{aligned}
$$

□



Client
0) Given a,b; f(a,b) := a ∧ b = ?
1) Choose keys Ka,Kb
2) $ENC_{Ka}(a) = a \oplus Ka$
$ENC_{Kb}(b) = b \oplus Kb$
13) $f(a,b) = DEC_{Kf}(ENC_{Kf}a \wedge b)) = a \wedge b$
↙ 3) $ENC_{Ka}(a)$, $ENC_{Kb}(b)$
↗ 11) $ENC_{Kf}(a \wedge b)$
↘ 3) Ka,Kb
↖ 12) Kf
Encrypted value holder
3) Choose random K2
4) $ENC_{Kb \oplus K2}(b) = ENC_{Kb}(b) \oplus K2$
6) $t0 := ENC_{Ka}(a) \wedge ENC_{Kb \oplus K2}(b)$
10) $ENC_{Kf}(a \wedge b) = t0 \oplus ENC_{K7}(p7) \oplus (K5 \wedge K6)$
→ 5) $ENC_{Kb \oplus K2}(b)$
← 8) K6
Keyholder
3) Choose random K6
4) $ENC_{K6}(Ka) = Ka \oplus K6$
6) $t2 := Ka \wedge ENC_{Kb \oplus K2}(b)$
10) $t3 := K6 \wedge ENC_{K5}(Kb \oplus K2)$
11) $Kf = t2 \oplus t3 \oplus K7$
↘ 5) $ENC_{Ka}(a)$,K2
↖ 9) $ENC_{K7}(p7)$,K5
↙ 5) Kb, $ENC_{K6}(Ka)$
↗ 9) $ENC_{K5}(Kb \oplus K2)$, K7
Helper
3) Choose random K5,K7
6) $ENC_{K5}(Kb \oplus K2) = Kb \oplus K2 \oplus K5$
7) $t1 := ENC_{Ka}(a) \wedge (Kb \oplus K2)$
8) $ENC_{K7}(p7) = t1 \oplus (ENC_{K6}(Ka) \wedge (Kb \oplus K2)) \oplus K7$

Figure 2: Algorithm for an AND ($\wedge$) operation of two bits using three parties.

6.1. Three Parties

It is possible to use only one helper, ie. three parties, for multiplication and AND. Let us show how to 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 1). 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 $K_b, ENC_{K_a}(a)$ and therefore they could reveal a secret. However, the remaining helper (ie. Helper 1) can compute on encrypted values, ie. the EVH can

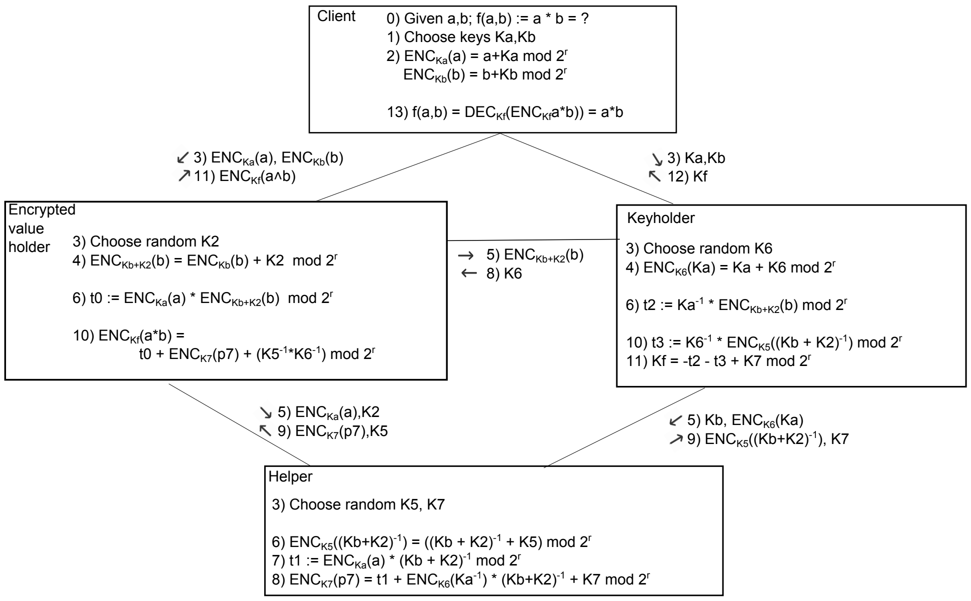

Client
0) Given a,b; f(a,b) := a * b = ?
1) Choose keys Ka,Kb
2) ENC_Ka(a) = a+Ka mod 2^r
ENC_Kb(b) = b+Kb mod 2^r
13) f(a,b) = DEC_Kf(ENC_Kf a*b)) = a*b
↙ 3) ENC_Ka(a), ENC_Kb(b)
↗ 11) ENC_Kf(a^b)
↘ 3) Ka,Kb
↖ 12) Kf
Encrypted value holder
3) Choose random K2
4) ENC_Kb+K2(b) = ENC_Kb(b) + K2 mod 2^r
6) t0 := ENC_Ka(a) * ENC_Kb+K2(b) mod 2^r
10) ENC_Kf(a*b) = t0 + ENC_K7(p7) + (K5^-1*K6^-1) mod 2^r
→ 5) ENC_Kb+K2(b)
← 8) K6
Keyholder
3) Choose random K6
4) ENC_K6(Ka) = Ka + K6 mod 2^r
6) t2 := Ka^-1 * ENC_Kb+K2(b) mod 2^r
10) t3 := K6^-1 * ENC_K5((Kb + K2)^-1) mod 2^r
11) Kf = -t2 - t3 + K7 mod 2^r
↘ 5) ENC_Ka(a),K2
↖ 9) ENC_K7(p7),K5
↙ 5) Kb, ENC_K6(Ka)
↗ 9) ENC_K5((Kb+K2)^-1), K7
Helper
3) Choose random K5, K7
6) ENC_K5((Kb+K2)^-1) = ((Kb + K2)^-1 + K5) mod 2^r
7) t1 := ENC_Ka(a) * (Kb + K2)^-1 mod 2^r
8) ENC_K7(p7) = t1 + ENC_K6(Ka^-1) * (Kb+K2)^-1 + K7 mod 2^r

Figure 3: Algorithm for multiplication of two integers using three parties.

encrypt $ENC_{K_b}(b)$ with a randomly chosen K_2 to obtain $ENC_{K_b \oplus K_2}(b)$. The helper can use $ENC_{K_b \oplus K_2}(b)$ instead of $ENC_{K_b}(b)$. 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 it does not obtain K_2) and the key $K2$ 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, ie. K_a with $K6$ and $K_b \oplus K_2$ with $K5$ and compute using Equation (3):

$$\begin{gathered} K_a \wedge (K_b \oplus K2) = \\ ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \\ \oplus K5 \wedge ENC_{K6}(K_a) \oplus K6 \wedge ENC_{K5}(K_b \oplus K_2) \oplus (K5 \wedge K6) \end{gathered} \tag{4}$$

In this case, we do not need to distribute all four terms to (four) different parties. In our scenario the (remaining) helper holds $K_b \oplus K_2$, $K5$ and $ENC_{K6}(K_a)$. Thus, it can compute two terms, namely $ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2)$ and $K5 \wedge ENC_{K6}(K_a)$. We can even simplify for the helper: $ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \oplus K5 \wedge ENC_{K6}(K_a) = ENC_{K6}(K_a) \wedge (K_b \oplus K_2)$.

This idea is realized in the protocol shown in Figure 2. The message complexity can be reduced by pre-sharing of keys. If, in addition, the client shares keys and secrets with all parties directly (rather than forwarding them using the KV and EVH), the helper receives all its messages in Step 5 directly from the client. Furthermore, the client is also assumed to have computed the second encryption of b, and a second encryption of K_b and shared it with KV, ie. $ENC_{K_b \oplus K2}(b)$, $ENC_{K5}(K_b \oplus K2)$. In this case only one communication round is necessary for a single multiplication, ie. only the helper must send a message to the EVH.

Theorem 2. *The AND protocol in Figure 2 is perfectly secure and correct.*

Proof. Analogously to the proof of Theorem 1, we show that decrypting encrypted result (Step 10 for the EVH) with the key (Step 11 for KH) in Figure 2 gives $a \wedge b$. We start by transforming $a \wedge b$ as shown below:

$$\begin{aligned}
a \wedge b =& \big((a \oplus K_a) \wedge (b \oplus (K_b \oplus K2))\big) \oplus \big((a \oplus K_a) \wedge (K_b \oplus K2)\big) \oplus \big(K_a \wedge (b \oplus (K_b \oplus K2))\big) \\
& \oplus \big(K_a \wedge (K_b \oplus K2)\big) \textit{ (Using Eq. 3)} \\
=& \big((a \oplus K_a) \wedge (b \oplus (K_b \oplus K2))\big) \oplus \big((a \oplus K_a) \wedge (K_b \oplus K2)\big) \\
& \oplus \big(K_a \wedge (b \oplus (K_b \oplus K2))\big) \\
& \oplus ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \oplus K5 \wedge ENC_{K6}(K_a) \\
& \oplus K6 \wedge ENC_{K5}(K_b \oplus K_2) \oplus (K5 \wedge K6) \textit{ (Using Eq. 4)} \\
=& ENC_{K_a}(a) \wedge ENC_{K_b \oplus K2}(b) \textit{(Using } ENC_K(m) = K \oplus m\textit{)} \\
& \oplus ENC_{K_a}(a) \wedge (K_b \oplus K2) \oplus K_a \wedge ENC_{K_b \oplus K2}(b) \\
& \oplus ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \\
& \oplus K5 \wedge ENC_{K6}(K_a) \oplus K6 \wedge ENC_{K5}(K_b \oplus K_2) \oplus (K5 \wedge K6) \\
=& t0 \oplus t1 \textit{ (Using } t0 := ENC_{K_a}(a) \wedge ENC_{K_b \oplus K2}(b), t1 := ENC_{K_a}(a) \wedge (K_b \oplus K2)\textit{)} \\
& \oplus t2 \textit{ (Using } t2 := K_a \wedge ENC_{K_b \oplus K2}(b)\textit{)} \\
& \oplus ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \\
& \oplus K5 \wedge ENC_{K6}(K_a) \oplus K6 \wedge ENC_{K5}(K_b \oplus K_2) \oplus (K5 \wedge K6) \\
=& t0 \oplus (K5 \wedge K6) \oplus t1 \oplus K7 \oplus K7 \\
& \oplus ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \oplus K5 \wedge ENC_{K6}(K_a) \\
& \oplus t2 \oplus K6 \wedge ENC_{K5}(K_b \oplus K_2) \textit{ (Rearranging and XOR with } K7 \oplus K7 = 0\textit{)} \\
=& t0 \oplus (K5 \wedge K6) \oplus t1 \oplus K7 \oplus K7 \oplus ENC_{K6}(K_a) \wedge (K_b \oplus K_2) \\
& \oplus t2 \oplus t3 \textit{ (Simplifying and using } t3 := K6 \wedge ENC_{K5}(K_b \oplus K_2)\textit{)} \\
=& t0 \oplus (K5 \wedge K6) \oplus t1 \oplus ENC_{K6}(K_a) \wedge (K_b \oplus K_2) \oplus K7 \oplus t2 \oplus t3 \oplus K7 \\
=& t0 \oplus (K5 \wedge K6) \oplus ENC_{K7}(p7) \oplus K7 \oplus \big(t2 \oplus t3 \oplus K7\big) \\
& \textit{(Using } ENC_{K7}(p7) = t1 \oplus ENC_{K6}(K_a) \wedge ENC_{K5}(K_b \oplus K_2) \oplus K7\textit{)} \\
=& t0 \oplus (K5 \wedge K6) \oplus ENC_{K7}(p7) \oplus K7 \oplus Kf \\
& \textit{(Using } Kf = t2 \oplus t3 \oplus K7\textit{, see Step 10, Figure 2)} \\
=& ENC_{Kf}(a \wedge b) \oplus Kf = a \wedge b \textit{ (Using Step 10, Figure 2)}
\end{aligned}$$

□

Theorem 3. *The multiplication protocol in Figure 3 is perfectly secure and correct.*

Proof. First, we express the product of a and b using encrypted values and keys:

$$\begin{aligned}
a \cdot b = &(a \cdot b) \mod 2^r \text{(by assumption)} \\
= &(a + ((K_a^{-1} + K_a) \mod 2^r) \cdot b) \mod 2^r \\
= &((a + K_a^{-1} + K_a) \cdot b) \mod 2^r \\
= &\big(\big((a + K_a) \cdot b\big) + \big(K_a^{-1} \cdot b\big)\big) \mod 2^r \\
= &\big(\big((a + K_a) \cdot (b + ((K_b^{-1} + K_b) \mod 2^r))\big) \\
&+ \big(K_a^{-1} \cdot (b + ((K_b^{-1} + K_b) \mod 2^r))\big)\big) \mod 2^r \\
= &\big(\big((a + K_a) \cdot (b + K_b^{-1} + K_b)\big) \\
&+ \big(K_a^{-1} \cdot (b + K_b^{-1} + K_b)\big)\big) \mod 2^r
\end{aligned} \tag{5}$$

We introduce using Equation (5) – analogous to Equation 4.

$$\begin{aligned}
&K_a^{-1} \cdot (K_b + K2)^{-1} \mod 2^r = \\
&ENC_{K6}(K_a^{-1}) \cdot ENC_{K5}((K_b + K_2)^{-1}) \\
&+ K5^{-1} \cdot ENC_{K6}(K_a^{-1}) \\
&+ K6^{-1} \cdot ENC_{K5}((K_b + K_2)^{-1}) + (K5^{-1} \cdot K6^{-1}) \mod 2^r
\end{aligned} \tag{6}$$

Next, we prove that decrypting encrypted result (Step 10 for the EVH) with the key (Step 11 for KF) in Figure 3 gives $a \cdot b$ as shown below:

$$\begin{aligned}
a \cdot b = &\big((a + K_a) \cdot (b + (K_b + K2))\big) + \big((a + K_a) \cdot (K_b + K2)^{-1}\big) + \big(K_a^{-1} \cdot (b + (K_b + K2))\big) \\
&+ (\big(K_a^{-1} \cdot (K_b + K2)^{-1}\big) \mod 2^r) \mod 2^r \textit{ (Using Eq. 5 and modulo laws)} \\
= &\big((a + K_a) \cdot (b + (K_b + K2))\big) + \big((a + K_a) \cdot (K_b + K2)^{-1}\big) \\
&+ \big(K_a^{-1} \cdot (b + (K_b + K2))\big) \\
&+ ENC_{K6}(K_a^{-1}) \cdot ENC_{K5}((K_b + K_2)^{-1}) + K5^{-1} \cdot ENC_{K6}(K_a^{-1}) \\
&+ K6^{-1} \cdot ENC_{K5}((K_b + K_2)^{-1}) + (K5^{-1} \cdot K6^{-1}) \mod 2^r \textit{ (Using Eq. 6)} \\
= &ENC_{K_a}(a) \cdot ENC_{K_b + K2}(b) \textit{(Using } ENC_K(m) = K + m \mod 2^r\textit{)} \\
&+ ENC_{K_a}(a) \cdot (K_b + K2)^{-1} + K_a^{-1} \cdot ENC_{K_b + K2}(b) \\
&+ ENC_{K6}(K_a^{-1}) \cdot ENC_{K5}((K_b + K_2)^{-1}) \\
&+ K5^{-1} \cdot ENC_{K6}(K_a^{-1}) + K6^{-1} \cdot ENC_{K5}((K_b + K_2)^{-1}) + (K5^{-1} \cdot K6^{-1}) \mod 2^r
\end{aligned}$$

$$
\begin{aligned}
&= t0 + t1 \textit{ (Using } t0 := ENC_{K_a}(a)\cdot ENC_{K_b+K2}(b), t1 := ENC_{K_a}(a)\cdot (K_b+K2)^{-1}\textit{)}\\
&\quad + t2 \textit{ (Using } t2 := K_a^{-1}\cdot ENC_{K_b+K2}(b)\textit{)}\\
&\quad + ENC_{K6}(K_a^{-1})\cdot ENC_{K5}((K_b+K_2)^{-1})\\
&\quad + K5^{-1}\cdot ENC_{K6}(K_a^{-1}) + K6^{-1}\cdot ENC_{K5}((K_b+K_2)^{-1}) + K5^{-1}\cdot K6^{-1} \mod 2^r\\
&= t0 + K5^{-1}\cdot K6^{-1} + t1 + ((K7^{-1}+K7) \mod 2^r)\\
&\quad + ENC_{K6}(K_a^{-1})\cdot ENC_{K5}((K_b+K_2)^{-1}) + K5^{-1}\cdot ENC_{K6}(K_a^{-1})\\
&\quad + t2 + K6^{-1}\cdot ENC_{K5}((K_b+K_2)^{-1}) \mod 2^r\\
&\quad \textit{(Rearranging and adding } ((K7^{-1}+K7) \mod 2^r)\textit{)}\\
&= t0 + (K5^{-1}\cdot K6^{-1}) + t1 + K7^{-1} + K7 + ENC_{K6}(K_a^{-1})\cdot (K_b+K_2)^{-1}\\
&\quad + t2 + t3 \mod 2^r \textit{ (Simplifying and using } t3 := K6^{-1}\cdot ENC_{K5}((K_b+K_2)^{-1})\textit{)}\\
&= t0 + (K5^{-1}\cdot K6^{-1}) + t1 + ENC_{K6}(K_a^{-1})\cdot (K_b+K_2)^{-1} + t2 + t3 + K7^{-1} \mod 2^r\\
&= t0 + (K5^{-1}\cdot K6^{-1}) + ENC_{K7}(p7) + \left(t2 + t3 + K7^{-1}\right) \mod 2^r\\
&\quad \textit{(Using } ENC_{K7}(p7) = t1 + ENC_{K6}(K_a)\cdot ENC_{K5^{-1}}((K_b+K_2)^{-1}) + K7\textit{)}\\
&= t0 + (K5^{-1}\cdot K6^{-1}) + ENC_{K7}(p7) + K7 + Kf^{-1} \mod 2^r\\
&\quad \textit{(Using } Kf^{-1} = (t2+t3+K7^{-1}) \mod 2^r\textit{, see Step 10, Figure 3)}\\
&= ENC_{Kf}(a\cdot b) + Kf^{-1} \mod 2^r = a\cdot b \textit{ (Using Step 10, Figure 3)}
\end{aligned}
$$

□

7. Re-Encryption and Arbitrary Expressions

In some cases two different encryptions of the same value might have to be used, eg. the KH and EVH can create them given a single encryption. To reencrypt a value a encrypted with key K_a. The KH chooses a key K'_a and transmits $ENC_{K'_a}(K_a^{-1})$ to the EVH, which computes $ENC_{ENC_{K'_a}(K_a^{-1})}(ENC_{K_a}(a)) = ENC_{K'_a}(a)$. 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: To compute $a \wedge b$ (see Figure 2), the helper gets the key K_b, the encrypted key $ENC_{K6}(K_a)$ 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 $ENC_{K_b}(b)$.

7.1. Large Fan-in

We can compute $a_0 \wedge a_1 \wedge \ldots \wedge a_{w-1}$ using two rounds only (Multiplication is analogous). We discuss two ways. We can trade the number of required (synchronized) rounds to exchange messages for employing more entities. More precisely, to AND w numbers rather than two numbers requires 2^w parties to get constant round complexity.

This follows from Equation (7) generalizing Equation (1), ie. there are 2^w terms with one term being computed by one entity. The second way uses only three parties. Each term in Equation (7) is an AND of encrypted values and keys. The EVH can compute the AND of all encrypted values locally and the KH can do the same for the keys. Each party encrypts its partial result and then they perform an AND of two partial results to obtain the term. This can be done for all terms in parallel.

Theorem 4. *An fan-gate $a_0 \wedge a_1 \wedge \ldots \wedge a_{w-1}$ can be evaluated in $O(1)$ 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). For a term consisting of w variables, each variable i can either be the key K_i of secret a_i or the encrypted value $ENC_{K_i}(a_i)$. Let S_w be all subsets of $\{0,1,...,w-1\}$. We have

$$\begin{aligned}
&a_0 \wedge a_1 \wedge \ldots \wedge a_{w-1} \\
=&(a_0 \oplus K_0 \oplus K_0) \wedge a_1 \wedge \ldots \wedge a_{w-1} \\
=&(ENC_{K_0}(a_0) \wedge a_1 \wedge \ldots \wedge a_{w-1}) \oplus (K_0 \wedge a_1 \wedge \ldots \wedge a_{w-1}) \\
=&\oplus_{S_E \in S_w} ((\wedge_{j \in S_E} ENC_{K_j}(a_j)) \wedge (\wedge_{j \in \{0,1,...,w-1\} \setminus S_E} K_j))
\end{aligned} \tag{7}$$

In the last step we applied to all a_i the same transformation as for a_0, ie. adding $K0 \oplus K0$ and expanding. We rearranged using the commutative property of $\wedge$.

In Equation (7) 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, eg. 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, ie. 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 2) to AND the two terms $t_E \wedge t_K$. They do this for all 2^w terms. The EVH computes the XORs of all encrypted resulted for all terms and the KH computes the XOR of all keys, which yields the final result for each party. □

8. Beyond Three Parties

For $n > 3$ parties we can ensure confidentiality given at most $n-2$ parties collude. The high level idea is that we always use a different set of three parties to compute a term contributing to the result. A classical result (Theorem 2 [7]) shows using two players the impossibility of collusion of $n/2$ parties. Note that we focus on a client-server setting, where parties do not share the result among each other. Furthermore, the actual computation is done through decomposition into terms that are XORed, such that a group of three parties computes a term. For any group of three parties, at most one party can be corrupt, which is below the $n/2$ threshold by [7]. We have one helper and one EVH and $n-2$ KHs, each holding a share of a key for an encrypted value.

Using more parties comes at a small price on the client side: The client must perform more operations (proportional to the number of parties that should not collude). The client encrypts each value m with $n-2$ keys $K_{m,i}$, ie. $ENC_{K_{m,0}\oplus K_{m,1}\oplus\ldots\oplus K_{m,n-3}}(m)$. We use $n-2$ KHs, such that KH i holds key $K_{m,i}$ for a value m, one EVH holding $ENC_{\oplus_{i\in[0,n-3]}K_{m,i}}(m)$ and one helper. We define $V_{m,i} := K_{m,i}$ for $i \in [0,n-3]$ and $V_{m,n-2} := ENC_{\oplus_{i\in[0,n-3]}K_{m,i}}(m)$. As in the three party case, XOR is performed by the EVH and the $n-2$ KHs by simply XORing their values. To AND two values we do an analogous derivation as for Equation (1):

$$\begin{aligned}
a \wedge b &= \big(a \oplus (K_{a,0} \oplus K_{a,1} \oplus \ldots K_{a,n-3}) \oplus (K_{a,0} \oplus K_{a,1} \oplus \ldots K_{a,n-3})\big)\\
&\wedge \big(b \oplus (K_{b,0} \oplus K_{b,1} \oplus \ldots K_{b,n-3}) \oplus (K_{b,0} \oplus K_{b,1} \oplus \ldots K_{b,n-3})\big)\\
&= \big(ENC_{\oplus_{i\in[0,n-3]}K_{a,i}}(a) \oplus (K_{a,0} \oplus K_{a,1} \oplus \ldots \oplus K_{a,n-3})\big)\\
&\wedge \big(ENC_{\oplus_{i=0}^{n-3}K_{b,i}}(b) \oplus (K_{b,0} \oplus K_{b,1} \oplus \ldots \oplus K_{b,n-3})\big)\\
&= \big(\oplus_{i=0}^{n-3} V_{a,i}\big) \wedge \big(\oplus_{i=0}^{n-3} V_{a,i}\big)\\
&= \oplus_{i,j=0}^{n-3}(V_{a,i} \wedge V_{b,j})\\
&= \oplus_{i,j=0}^{n-3}(K_{(ab),(i,j)} \oplus K_{(ab),(i,j)}) \oplus_{i,j=0}^{n-3} (V_{a,i} \wedge V_{b,j})\\
&= \big(\oplus_{i,j=0}^{n-3} K_{(ab),(i,j)}\big) \oplus \big(\oplus_{i,j=0}^{n-3} \big(V_{a,i} \wedge V_{b,j} \oplus K_{(ab),(i,j)}\big)\big)\\
&= \big(\oplus_{i,j=0}^{n-3} K_{(ab),(i,j)}\big) \oplus \big(\oplus_{i,j=0}^{n-3} \big(ENC_{K_{(ab),(i,j)}}(V_{a,i} \wedge V_{b,j})\big)\big) \qquad (8)
\end{aligned}$$

There are $(n-1)^2$ terms in Equation (8) in contrast to four terms in Equation (1). Next we show that only one helper interacting with all the $n-2$ KHs and the EVH is enough. Initially, each party P_i has two values $V_{a,i}$ and $V_{b,i}$. Say we want to compute a term $V_{a,i} \wedge V_{b,j}$. The party P_i having $V_{a,i}$ and the party P_j having $V_{b,j}$ can compute the AND using the helper. One party (either P_i or P_j) takes the role of the EVH and the other the role of the KH in the AND protocol (Figure 2). They treat $V_{a,i}$ and $V_{b,j}$ as secret, ie. each party encrypts its secret $V_{a,i}$ and $V_{b,j}$ to get $ENC_{K_{V_{a,i}}}(V_{a,i})$ and $ENC_{K_{V_{b,j}}}(V_{b,j})$ and distributes the encrypted values and keys before running the protocol in Figure 2. This yields an outcome of the form $ENC_{K_{(ab),(i,j)}}(V_{a,i} \wedge V_{b,j})$ for the party taking the role of the EVH in the AND protocol (Figure 2) and $K_{(ab),(i,j)}$ for the KH in the AND protocol. Each party, can then locally aggregate its results by XORing them.

Theorem 5. *Using n parties up to $n-2$ parties can collude while maintaining data confidentiality.*

Proof. Clearly, if initially $n-2$ party collude this is not a problem, since each party P_i holds only two values $V_{a,i}$ and $V_{b,i}$. Thus, $n-2$ values do not allow to get all $n-2$ keys as well as the encrypted value. To disclose a,b or $a \wedge b$ a party needs all parts, ie. $V_{a,i}$ to decrypt a, $V_{a,j}$ to decrypt b, $V_{a,j} \wedge V_{b,i}$ to decrypt $a \wedge b$. Consider the computation of an arbitrary ANDed $V_{a,i} \wedge V_{b,j}$ term in Equation (8). If at least two parties out of the three parties P_i, P_j and helper are corrupt, the shares $V_{a,i}$ and $V_{b,i}$ are disclosed. If there is at most one corrupt party then no information is disclosed due to Theorem 2. Thus,

Operation	GRR [ys/op/party]	Our scheme [ys/op/party]	SpeedUp	Plaintext [ys/op]	Overheadfactor (of our scheme)
Encryption	6.7600	0.0078	866.7	-	-
Decryption	9.7233	0.0011	9115.6	-	-
Addition	0.0655	0.0015	42.3	0.0015	1
Multiplication	4.0600	0.0150	270.67	0.0031	4.84

Table 1: Comparison of local computation of the slowest party.

Operation	[total transmitted bits (rounds)]				
	Our scheme	GRR, BGW	Sharemind	[24]	[3]
Addition	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Multiplication	160 (2)	390 (1)	$\geq$ 500 (3)	390 (1)	$>$ 300($>$ 2)
	Our scheme	GMW	BMR		
XOR	0 (0)	0 (0)	>0 (0)		
AND	5 (2)	>50 ($>$ 2)	$>$ 10 ($>$ 2)		

Table 2: Comparison of total communication and rounds.

the helper does also not learn anything about $V_{a,i}$ or $V_{b,j}$. Therefore, also neither P_i learnt anything about $V_{b,j}$ nor did P_j learn anything about $V_{a,i}$ or the term $V_{a,i} \wedge V_{b,j}$. By assumption there at most $n-2$ corrupt parties, assume w.l.o.g. that P_0 and P_1 are not corrupt. Then the shares $V_{a,0}, V_{a,1}, V_{b,0}$ and $V_{b,i}$ as well as their ANDs, eg. $V_{a,0} \wedge V_{b,0}$, are not disclosed. However, since all shares are needed to disclose any information the computation is secure. □

9. (Empirical) Evaluation

The paper focuses on the foundations of the JOS scheme. But we provide a small evaluation with the intention to measure the overhead for encryptions, decryptions, additions and multiplications of 32bit unsigned integers compared to GRR in terms of computation and communication. We implemented GRR relying on Shamir's Secret Sharing [23] in C++ using the NTL lib. We ran our experiments on an Intel Core i5. For GRR we used a 65 bit prime, since to support multiplications of two 32 bit numbers, GRR needs at least a prime of 65 bits.

To assess computational performance we computed the average time per operation per party in microseconds using 10 million operations, denoted by '[ys/op/party]' in Table 1. Whereas for GRR all parties perform the same computations, in our scheme there are differences. We used the slowest party to make the comparison fair. Our scheme outperforms GRR due to several reasons. Our scheme does not require the use of the NTL lib and needs only additions and multiplications. GRR needs also more expensive operations such as modulo, ie. divisions. Encryption and decryption are extremely efficient in our scheme, since we require essentially only one addition or subtraction (and generation of the key). GRR requires generation of several random

numbers for encryption and evaluation of a polynomial (for encryption). Decryption requires solving a system of equations.
For GRR secrets are hosted at the parties. Therefore, we also assumed that keys and secrets are held at the parties, ie. they do not have to be distributed by the client to compare our protocol (Multiplication is analogous to AND in Figure 2. This means that all messages to and from the client are not considered. The amount of communication to perform one addition or one multiplication is given in Table 2. Table 2 is based on pre-shared keys between each pair of parties, eg. by exchanging a single key and then using this key as input for a pseudo random number generator. Thus, we need two rounds and we have to transmit the analogous terms to $ENC_{Ka}(a), ENC_{Kb}(b), ENC_{Kb\oplus K2}(b), ENC_{K7}(p7)$, $ENC_{K5}(Kb \oplus K2)$ in Figure 2 (requiring 32 bits each). For GRR each party sends two messages with 65 bits each. We also compared against schemes that provide only statistical security, ie. for [24] we chose a keysize of size 64 bits, which is generally not enough. For GMW [18] we assumed that the security parameter used in oblivious transfer is 64 bits, which is also very modest. For BMR [4] we also lower bounded the number of bits exchanged. Table 2 indicates that our scheme uses least exchanged bits. With respect to the number of rounds GRR (and Maurer) do better. For the important case, where we do big data analysis, the bottleneck becomes bandwidth, ie. the amount of communicated information.

10. Conclusions

We have presented a new methodology for secure-multi party computation. It strikes through little communication, storage and computational overhead, being unconditionally secure and conceptual simplicity. This makes it easily implementable and a suitable candidate for a framework targeted towards privacy preserving data mining, where minimal amount of communication as provided by JOS is essential. Thus, in future work, we want to implement an industrial strength system focusing on numerical operations.

11. References

[1] G. Asharov and Y. Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, pages 1–94, 2011.

[2] 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.

[3] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, 1989.

[4] 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.

[5] A. Beimel. Secret-sharing schemes: a survey. In *Coding and cryptology*, pages 11–46. Springer, 2011.

[6] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 478–492. IEEE, 2013.

[7] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, 1988.

[8] 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.

[9] D. Boneh and M. Franklin. Efficient generation of shared rsa keys. In *Advances in Cryptology (CRYPTO)*, pages 425–439. 1997.

[10] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks*. 2010.

[11] 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.

[12] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology-CRYPTO 2007*, pages 572–590. Springer, 2007.

[13] I. Damgård and S. Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography*, pages 621–641. Springer, 2013.

[14] 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.

[15] 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.

[16] 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.

[17] C. Gentry, S. Gorbunov, S. Halevi, V. Vaikuntanathan, and D. Vinayagamurthy. How to compress (reusable) garbled circuits. *IACR Cryptology ePrint Archive*, 2013:687, 2013.

[18] 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.

[19] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography*, pages 294–314. Springer, 2009.

[20] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

[21] J. Launchbury, D. Archer, T. DuBuisson, and E. Mertens. Application-scale secure multiparty computation. In *Programming Languages and Systems*, pages 8–26. Springer, 2014.

[22] P. Lory. Secure distributed multiplication of two polynomially shared values: Enhancing the efficiency of the protocol. In *Emerging Security Information, Systems and Technologies (SECURWARE)*, pages 286–291, 2009.

[23] P. Lory and J. Wenzl. A note on secure multiparty multiplication. *ItemID 20267, University of Regensburg*, 2011.

[24] U. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[25] 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.

[26] J. Schneider. Secure numerical and logical multi party operations. *arXiv preprint arXiv:1511.03829,* `http://arxiv.org/abs/1511.03829`, 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] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science(FOCS)*, 1986.