From oblivious AES to efficient and secure database join in the multiparty setting

Sven Laur[2,3], Riivo Talviste[1,2]⋆, and Jan Willemson[1,3]⋆⋆

[1] Cybernetica, Ülikooli 2, Tartu, Estonia
[2] Institute of Computer Science, University of Tartu, Liivi 2, Tartu, Estonia
[3] Software Technology and Applications Competence Center, Ülikooli 2, Tartu, Estonia

Abstract. AES block cipher is an important cryptographic primitive with many applications. In this work, we describe how to efficiently implement the AES-128 block cipher in the multiparty setting where the key and the plaintext are both in a secret-shared form. In particular, we study several approaches for AES S-box substitution based on oblivious table lookup and circuit evaluation. Given this secure AES implementation, we build a universally composable database join operation for secret shared tables. The resulting protocol scales almost linearly with the database size and can join medium sized databases with $100,000$ rows in few minutes, which makes many privacy-preserving data mining algorithms feasible in practice. All the practical implementations and performance measurements are done on the Sharemind secure multi-party computation platform.

1 Introduction

Many information systems need to store and process private data. Encryption is one of the best ways to assure confidentiality, as it is impossible to learn anything from encrypted data without knowledge of the private key. However, the number of processing steps one can carry out on encrypted data is rather limited unless we use fully homomorphic encryption. Unfortunately, such encryption schemes are far from being practical even for moderate-sized data sets [21].

Another compelling alternative is share-computing, since it assures data confidentiality and provides a way to compute on secret shared data, which is several magnitudes more efficient than fully homomorphic encryption. In this setting, data is securely shared among several parties so that individual parties learn

⋆ This research was supported by European Social Fund Doctoral Studies and Internationalisation Programme DoRa.

⋆⋆ This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of Defense or the U.S. Government. The research complied with all relevant U.S. Government data privacy protections. Distribution Statement A (Approved for Public Release, Distribution Unlimited).

nothing about shared values during the computations and the final publication of output shares reveals only the desired output(s). For most share-computing systems, even a coalition of parties cannot learn anything about private data unless the size of a coalition is over a threshold.

Development and implementation of such multi-party computing platforms is an active research area. FairPlayMP [6], SecureSCM [2], SEPIA [13], Sharemind [8], VMCrypt [30] and TASTY [24] computing platforms represent only some of the most efficient implementations and share-computing has been successfully applied to real-world settings [10,9].

Note that various database operations are particularly important in privacy-preserving data processing. Efficient and secure protocols for most key operations on secret-shared databases are already known, see [29]. The most notable operation still missing is database join based on secret-shared key columns. This operation can be used e.g. for combining customer data coming from different organisations or linking the results of statistical polls into a single dataset.

Our main theoretical contribution is an efficient multi-party protocol for database join, which combines oblivious shuffle with pseudorandom function evaluation on secret-shared data. In practice, we instantiate the pseudorandom function with the AES-128 block cipher and implement it on the Sharemind platform [8]. The latter is a non-trivial task, since the input and the secret key are secret-shared in this context. The resulting AES-evaluation protocol is interesting in its own right. First, AES is becoming a standard performance benchmark for share-computing platforms [18,25,33,28] and thus we can directly compare how well the implementation on the Sharemind platform does. Second, a secret-shared version of AES can be used to reduce security requirements put onto the key management of symmetric encryption [18]. In brief, we can emulate trusted hardware encryption in the cloud by sharing a secret key among several servers.

2 Preliminaries

AES. Advanced Encryption Standard (AES) is a symmetric block cipher approved by the National Institute of Standards and Technology [31]. AES takes a 128-bit block of plaintext and outputs 128 bits of corresponding ciphertext. AES can use cipher keys with lengths of 128, 192 or 256 bits. In our work we will only use AES-128, which denotes AES with 128-bit keys.

Sharemind platform. Sharemind platform is a practical and secure share-computing framework for privacy-preserving computations [8], where the private data is shared among three parties referred to as *miners*. In its original implementation, Sharemind uses additive secret sharing on 32-bit integers, i.e., a secret s is split into three shares s_1, s_2, s_3 such that $s = s_1 + s_2 + s_3 \bmod 2^{32}$. In this work, we use bitwise sharing where the secret can be reconstructed by XOR-ing individual shares: $s = s_1 \oplus s_2 \oplus s_3$.

The current Sharemind implementation is guaranteed to be secure only if the adversary can observe the internal state of a single miner node. Thus, we report performance results only for the *semi-honest setting*. Additionally, we show how to generalise our approach to malicious setting. The latter is rather straightforward, as all protocols are based only on secure addition and multiplication protocols. Although the bitwise sharing alone is not secure against *malicious corruption*, shared message authentication codes can be used to guarantee integrity of secret sharings throughout the computations [19,32].

Security definitions and proofs. We use standard security definitions based on ideal versus real world paradigm. In brief, security is defined by comparing a real protocol with an ideal implementation where a trusted third party privately collects all inputs, does all computations and distributes outputs to corresponding parties. We say that a protocol is secure if any plausible attack against real protocol can be converted to an attack against ideal protocol such that both attacks have comparable resource consumption and roughly the same success rate, see standard treatments [22,14,15] for further details.

A canonical security proof uses a wrapper (*simulator*) to link a real world adversary with the ideal world execution model. More precisely, the simulator has to correctly fake missing protocol messages and communicate with the trusted party. As most protocols are modularly built from sub-protocols, security proofs can be further compacted. Namely, if all sub-protocols are *universally composable*, then we can prove the security in the hybrid model where executions of all sub-protocols are replaced with ideal implementations [15].

Since almost all share-computing platforms including Sharemind provide universally composable data manipulation operations, we use this composability theorem to omit unnecessary details from security proofs (see also [8]).

Efficiency metrics in protocol design. Real-life efficiency of a protocol execution depends on the number of rounds and the total amount of messages sent over communication channels. The actual dependency is too complicated to analyse directly. Hence, we consider two important sub-cases. When the total communication is small compared to channel bandwidths, then the running time depends linearly on the number of rounds. If the opposite holds, then running time depends linearly on the communication complexity.

3 Share-computing protocol for AES block cipher

The overall structure of our protocol follows the standard AES algorithm specification [31]. However, there are some important differences stemming from the fact that the secret key and the message is bitwise secret shared and we have to use share-computing techniques. Fortunately, three out of four sub-operations are linear and thus can be implemented by doing local share manipulations. The efficiency of the AES protocol implementation is determined by `SubWord()`

and `SubBytes()` operations that evaluate the S-box on secret-shared data. The `SubWord()` function used in key expansion applies the S-box independently to each byte of its input word. Similarly, the `SubBytes()` function uses the S-box independently on each byte of the 4-word state given as the argument.

3.1 S-box evaluation protocol based on oblivious selection

As the AES S-box is a non-linear one-to-one mapping of byte values, it can be implemented as 256 element lookup table. In our setting, the input of the S-box is secret shared and we need oblivious array selection to get the shares of the right table entry. The latter can be achieved by using various techniques from [29]. First, we must convert the input x into a zero-one index vector $\boldsymbol{z}$ where all entries, except one, are zeros. The non-zero vector element z_x corresponds to the entry in the S-box array that we want to pick as the output. More precisely, let $x_7x_6\dots x_0$ be the bit-representation of the input x and $i_7i_6\dots i_0$ be the bit-representation of an index i. Then $z_i = [x_7 = i_7] \wedge \dots \wedge [x_0 = i_0]$ and the shares of index vector $\boldsymbol{z}$ can be computed by evaluating multinomials

$$z_i = (x_7 \oplus i_7 \oplus 1) \cdots (x_0 \oplus i_0 \oplus 1) \ . \tag{1}$$

For example, the first entry can be computed as $z_0 = (1 \oplus x_7)(1 \oplus x_6)\dots(1 \oplus x_0)$ and the second entry as $z_1 = (1 \oplus x_7)(1 \oplus x_6)\dots(1 \oplus x_1)x_0$.

Note that each multinomial z_i is of of degree 8 and thus 1792 secure multiplications over $\mathbb{F}_2$ are needed. To reduce the number of communication rounds, we gather terms $b_{ij} = x_j \oplus i_j \oplus 1$ into eight 256 element vectors:

$$\boldsymbol{b}_7 = (b_{0,7}, \dots, b_{255,7}), \dots, \boldsymbol{b}_0 = (b_{0,0}, \dots, b_{255,0})$$

and use vectorised bitwise multiplications to multiply all eight terms in the same row. If we do them sequentially, then the computation of index vector requires seven multiplication rounds. With tree-style evaluation strategy we can reduce the number of multiplication rounds to three. For that, we must evaluate same level brackets in parallel for $\boldsymbol{z} = ((\boldsymbol{b}_7 \cdot \boldsymbol{b}_6) \cdot (\boldsymbol{b}_5 \cdot \boldsymbol{b}_4)) \cdot ((\boldsymbol{b}_3 \cdot \boldsymbol{b}_2) \cdot (\boldsymbol{b}_1 \cdot \boldsymbol{b}_0))$. The multiplicative complexity of this step can be further decreased by utilising the underlying recursive structure of the index vector, as proposed by Launchbury *et al.* [28]. For comparison, we also reimplemented their solution.

As the second step, we must compute scalar product between the indicator vector $\boldsymbol{z}$ and 256-element output table $\boldsymbol{y}$ of the S-box. As elements of $\boldsymbol{y}$ are 8-bit long whereas elements of $\boldsymbol{z}$ are from $\mathbb{F}_2$, we must select output bits one by one. Let $\boldsymbol{y}_j = (y_{0,j}, \dots, y_{255,j})$ denote the vector of jth bits in the output table $\boldsymbol{y}$. Then the jth output bit f_j of the S-box can be computed as

$$f_j = \langle \boldsymbol{z}, \boldsymbol{y}_j \rangle = \sum_{i=0}^{255} z_i \ \ y_{ij} \tag{2}$$

over $\mathbb{F}_2$. Since the output table $\boldsymbol{y}$ is public, all operations can be done locally and the second step does not contribute to the communication complexity.

3.2 S-box evaluation protocol based on circuit evaluation

The oblivious indexing as a generic approach is bound to provide a protocol with sub-optimal multiplication complexity, as the two stage evaluation of output bits f_j forces us to compute terms z_i that are dropped in the equation (2).

We can address this issue by secure computation techniques based on branching programs [26]. For that, we must convert the expression for f_j into a binary decision diagram $\mathcal{B}$ with minimal number of decision nodes. After that we must build a corresponding arithmetic circuit that evaluates $\mathcal{B}$ in bottom-up manner. As each decision node introduces two secure multiplications, the efficiency of the resulting protocol is determined by the shape of $\mathcal{B}$. Let c denote the total number of decision nodes and d denote the longest path in $\mathcal{B}$. Then the resulting protocol consists of $2c$ secure multiplication operations over $\mathbb{F}_2$, which can be arranged into d rounds of parallel multiplications.

Although this approach produces significant gains, we can use recent findings in hardware optimisation to boost efficiency further. Circuit minimisation for the AES S-box is a widely studied problem in the hardware design with many known results. In this work, we use the designs by Boyar and Peralta [11,12]. Note that their aim was to minimise the total number of gates and the overall circuit depth, while we need a circuit with minimal number of multiplication gates (AND operations) and with paths that contain as few multiplications as possible, i.e., have a low multiplicative depth. Hence, their best design with 128 gates is not the best for our purposes, as it contains 34 multiplications and its multiplicative depth is 4, while their older design [11] contains 32 multiplication and has a multiplicative circuit depth 6. Of course, the multiplicative depth plays also important role in the protocol, when the bandwidth is high, hence, the newer design might have advantages when only a few AES evaluations are performed.

As extended versions of both articles contain straight-line C-like programs for their circuits, it is straightforward to implement the corresponding secure evaluation protocol with a minor technical tweak. As byte is the smallest data unit supported by network communication libraries, entire byte is used to send elements of $\mathbb{F}_2$ over the network during a secure multiplication protocol. We can eliminate this bloat by doing eight multiplications in parallel, since eight individual values can be packed into the same byte.

It is straightforward to achieve this grouping for the `SubBytes()` function, as it evaluates 16 S-boxes in parallel. Consequently, if we treat original variables as 16-element bit-vectors, we can evaluate all 16 copies of the original circuit in parallel without altering the straight-line program. For the `SubWord()` function, additional regrouping is necessary, as it evaluates only four S-boxes in parallel. It is sufficient if we must split all multiplications into pairs that can be executed simultaneously so that we can do eight multiplications in parallel.

3.3 Security analysis for the entire protocol

Note that all three versions of the AES S-box evaluation algorithms are arithmetic circuits consisting of addition and multiplication gates. Hence, it is straightforward to prove the following result.

Theorem 1. *If a share-computing framework provides universally composable protocols for bitwise addition, bitwise multiplication and bit decomposition, then all three AES S-box implementations are universally composable. Any universally composable AES S-box implementation gives a rise to a universally composable share-computing protocol for the AES block cipher.*

Proof. The proof follows directly from the universal composability theorem as we use share-computing protocols to evaluate arithmetic circuits. □

Note that this result holds for any corruption model including the Sharemind framework, which provides security against one-out-three static passive corruption. To get security against active corruption, the underlying secret sharing scheme must support both bitwise addition and multiplication while being verifiable. There are two principal ways to achieve this.

First, we can embed elements of $\mathbb{F}_2$ into some larger finite field $\mathbb{F}_{2^t}$ with extension element α and then use standard verifiable secret sharing schemes which support secure multiplication over $\mathbb{F}_{2^t}$. On top of that it is rather straightforward to implement universally composable bit decomposition [17], which splits a secret $x \in \mathbb{F}_{2^t}$ into a vector of shared secrets $x_{t-1}, \ldots, x_0$ such that $x = x_{t-1}\alpha^{t-1} + \cdots x_1\alpha + x_0$. As a consequence, all three assumptions of Theorem 1 are satisfied and we get a secure protocol for evaluating AES. However, there is a significant slowdown in the communication due to prolonged shares.

Alternatively, we can use oblivious message authentication [19] to protect individual bits without extending shares. However, this step attaches a long secret shared authentication code to each bit. To avoid slowdown, we can authenticate long bit vectors with a singe authentication code. The latter fits nicely into the picture, as we have to evaluate 16 circuits in parallel.

3.4 Further tweaks of the AES evaluation protocol

Block ciphers are often used to encrypt many messages under the same secret key. In such settings, it is advantageous to encrypt several messages in parallel in order to reduce the number of communication rounds. The latter is straightforward in the Sharemind platform, as it naturally supports parallel operations with vectors. The corresponding vectorised AES protocol takes in a vector of plaintext shares and a vector of shared keys and outputs a vector of cipher text shares. As another efficiency tweak note that we need to execute that key scheduling only once if the secret key is fixed during the encryption. Hence,we can run the key scheduling protocol separately and store the resulting shares of all 128-bit round keys for later use. The corresponding separation of pre-processing and online phases decreases amortised complexity by a fair margin.

Protocol	Multiplicative depth	Running time (1 evaluation)	Multiplicative complexity	Running time (4096 evaluations)
ObSel	3	32.5 ms	1792	9051 ms
Lddam	3	**31.1** ms	304	1109 ms
BCirc-1	6	69.6 ms	32	148 ms
BCirc-2	4	40.8 ms	34	**127** ms

Table 1. Performance results of various S-box evaluation algorithms.

3.5 Efficiency metrics and real-life performance

Having established essentially four methods with very different complexity parameters, we need to compare their real-life performance. For that we have implemented four versions of `SubBytes()` routines on the Sharemind platform and measured the actual performance. The tests were done on a cluster where each of the three Sharemind miners was deployed in a separate machine. The computers in the cluster were connected by an ethernet local area network with link speed of 1 Gbps. Each computer in the cluster had 48 GB of RAM and a 12-core 3 GHz CPU with Hyper Threading. The channels between the computers were also encrypted using 256-bit elliptic curve key agreement and the ChaCha stream cipher [7] provided by the underlying RakNet networking library [1]. While the choice of ChaCha is not standard, the best known attacks against it are still infeasible in practice [5].

We considered algorithms in two different settings. First, we measured the time needed to complete a single evaluation of `SubBytes()` function. Second, we measured how much time does it take to evaluate 4096 `SubBytes()` calls in parallel. The first setting corresponds to the case where various delays have dominant impact on the running-time, whereas the effect of communication complexity dominates in the second case. Table 1 compares theoretical indicators[4] and practical performance for all four protocols. The ObSel protocol is based on oblivious selection vector and Lddam is the same protocol with reduced number of multiplications [28]. Protocols based on Boolean circuits designed by Boyar and Peralta are denoted by BCirc-1, BCirc-2.

The results clearly show that multiplicative depth and complexity are good theoretical performance measures for optimising the structure of arithmetic circuits, as they allow us to predict the running times with $10 - 20\%$ precision. Each communication round costs $10 - 12$ ms in single operation mode and each multiplication operation adds $3.5 - 5.1$ ms to amortised running-time.

Secondly, we measured amortised cost of the AES evaluation protocol with precomputed round keys, see Figure 1. As expected, various algorithms have different saturation points where further parallelisation does not decrease the amortised cost any more. In particular, note that for few blocks the amortised costs of Lddam and circuit evaluation algorithms BCirc-1 and BCirc-2 is

[4] As all multiplications are carried over $\mathbb{F}_2$, we do not have to compensate for various input lengths and can just count the number of multiplications.

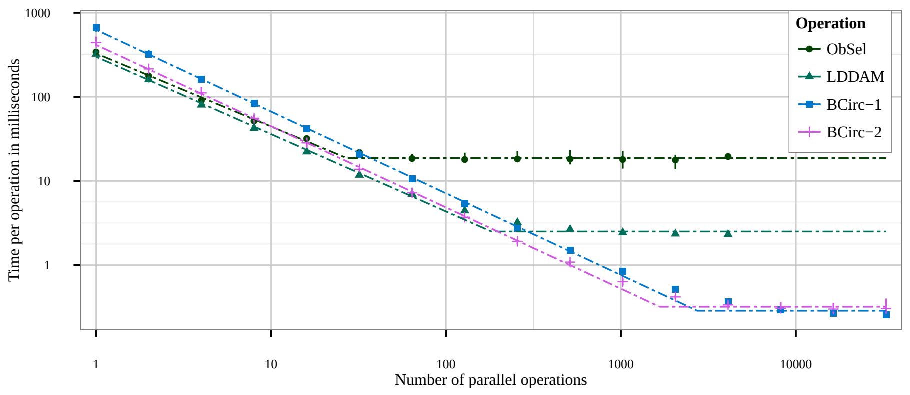


Fig. 1. Performance of AES evaluation protocols using precomputed round keys.

comparable, i.e., the advantage of circuit evaluation manifests only if we encrypt around 80 plaintexts in parallel. Also, note that the newer design BCirc-2 with smaller multiplicative depth performs better when the number of encryption calls is between $100 - 10,000$. After that the impact of communication complexity becomes more prevalent and the BCirc-1 protocols becomes more efficient.

As the final test, we measured the running time of the AES protocol with and without key scheduling. Table 2 depicts the corresponding results. As before, we give the running times for a single encryption operation and limiting cost of a single operation if many encryptions are done in parallel. Mode I denotes encryption with key expansion and mode II denotes encryption with pre-expanded secret key. Again, the results are in good correspondence. The cost of a single operation is roughly two times slower with the key expansion[5] , since computing a shared round key requires one parallel invocation of S-boxes. For the amortised cost, the theoretical speedup should be 1.25 as there are 20 S-box invocation per round in the mode I and 16 invocations per round in the mode II. The difference in actual speedup factors suggest existence of some additional bottlenecks in our key-expansion algorithms.

Table 3 compares our results with the state of the art in oblivious AES-128 evaluation protocols. To make results comparable, the table contains results only for the semi-honest setting. In most cases, authors report the performance of AES with pre-shared keys (mode II). More than tenfold difference between two-party and three-party implementations is expected, as two-party computations require costly asymmetric primitives. Note that the cost of single operation for our implementation in Table 3 uses the approach of Launchbury *et al.*, whereas the amortized time is obtained using the circuit-based approach.

[5] The slowdown can be further reduced to 1.2 if we compute next subkey in parallel with the AES round to reduce multiplicative depth of the circuit.

	Single operation			Amortised cost		
	Mode I	Mode II	Ratio	Mode I	Mode II	Ratio
ObSel	682 ms	343 ms	1.99	20.34 ms	18.69 ms	1.09
Lddam	**652** ms	**323** ms	2.02	4.16 ms	2.51 ms	1.66
BCirc-1	1329 ms	664 ms	2.00	0.48 ms	**0.29** ms	1.68
BCirc-2	890 ms	443 ms	2.01	**0.37** ms	0.32 ms	1.17

Table 2. Performace results for various AES evaluation algorithms

We can not fully explain roughly 20 times performance difference between the two implementations of single operation following the approach of Launchbury *et al.* Possible explanations include measurement error and extreme concentration on the network layer optimization by the authors of [28].

Authors	Reference	Setting	Mode	Single operation	Amortised cost
Pinkas et al.	[33]	2-party	II	5000 ms	— ms
Huang et al.	[25]	2-party	II	200 ms	— ms
Damgård and Keller	[18]	3-party	I	2000 ms	— ms
Launchbury et al.	[28]	3-party	II	**14.28** ms	3.10 ms
This work		3-party	II	323 ms	**0.29** ms

Table 3. Comparison of various secure AES-128 implementations

4 Secure database join

As mentioned in the introduction, secure database join is a way to combine several data sources in privacy-preserving manner. In this work, we consider the most commonly used *equi-join*[6] operation, which merges tables according to one of few key columns using the equality comparison in the join predicate. In many cases, the key value is unique, such as social security number or name and postal code combined. The uniqueness assumption significantly simplifies our task. The need to deal with the colliding keys significantly increases the complexity of the protocols, and this case is handled in Section 4.3.

An ideal secure inner join protocol takes two or more secret-shared database tables and produces a new randomly ordered secret-shared table that contains the combined rows where the join predicate holds. The parties should learn nothing except for the number of rows in the new database. The random reordering of the output table is necessary to avoid unexpected information propagation when some entries are published either for input or for the output table.

[6] The authors adapt the Structured Query Language (SQL) terminology in this paper.

Let m_1 and m_2 denote the number of rows and n_1 and n_2 the number of columns in the input tables. Then it is straightforward to come up with a solution that uses $\Theta(m_1 m_2)$ oblivious comparison operations by mimicking a naïve database join algorithm. We can obliviously compare all the possible key pairs, shuffle the whole database, open the comparison column and remove all the rows with the equality bit set to 0. It is straightforward to prove that this protocol is secure, since it mimics the actions of ideal implementation in verbatim. We will refer to this algorithm as NaiveJoin and treat it as a baseline solution.

Theorem 2. *If a share-computing framework provides universally composable protocols for database shuffle and oblivious comparison, then the* NaiveJoin *protocol is universally composable in the information theoretical model.*

Proof (Sketch). The formal proof hinges on three facts. First, the oblivious comparison leaks no information and the extra column can be simulated without problems. Second, the shuffle completely hides the order of the database rows. Third, the number of rows can be deduced from the ideal output and it coincides with the number of ones in the publishing phase.

In the simulation construction, we fake all shares until the shuffle phase without any knowledge of the true output. Next, we use share shuffle protocol to extract input shares of all malicious parties and submit them to the trusted third party who outputs the corresponding output shares. The simulator assigns them to random rows for $m_1 m_2$ shuffle outputs and sets the corresponding comparison column to 1. Remaining rows are filled with shares of zeroes. After that the simulator continues with the shuffle simulation with this shuffle output table. Publishing of shares is done according to the shuffle output table. □

4.1 Secure inner join based on unique key column

As the first step towards a more efficient algorithm, consider a setting where the computing parties (miners) obliviously apply pseudorandom permutation π_s to encrypt the key column. As π_s is a pseudorandom permutation (a block cipher depending on an unknown key s) and all the values in the key column are unique, the resulting values look completely random if none of the miners knows π_s. Hence, it is secure to publish all the encryptions of key columns. Moreover, the tables can be correctly joined using the encryptions instead of key values.

However, such a join still leaks some information – miners learn which database rows in the first table correspond to the database rows in the second table. By shuffling the rows of initial tables, this linking information is destroyed. The resulting algorithm is depicted as Protocol 1. We emphasise that in each step all the tables are in secret-shared form. In particular, each miner carries out step 4 with its local shares and thus the table T^* is created in a secret-shared form. Note that we have also added step 5 to deal with the case of colliding keys. This case will be discussed in Section 4.3.

As the actual join operation is performed on public (encrypted) values, the construction works also for the *left* and *right outer joins*, where either the left or

Database shuffling phase

1. Miners obliviously shuffle each database table T_i.
   Let T_i^* denote the resulting shuffled table with a key column $\boldsymbol{k}_i^*$.

Encryption and join phase

2. Miners choose a pseudorandom permutation π_s by generating a shared key s.
3. Miners obliviously evaluate π_s on all shared key columns $\boldsymbol{k}_i^*$.
4. Miners publish all values $\pi_s(k_{ij}^*)$ and use standard database join to merge the tables based on columns $\pi_s(\boldsymbol{k}_i^*)$. Let T^* be the resulting table.

Optional post-processing phase for colliding keys

5. If there are some non-unique keys in some key column $\pi_s(\boldsymbol{k}_i^*)$, miners should perform additional oblivious shuffle on the secret-shared table T^*

Protocol 1: Secure implementation of PrpJoin operation

right table retains all its rows, whether a row with a matching key exists in the other table or not. These outer joins are common in data analysis. For instance, given access to supermarket purchases and demographic data, we can use outer join to add person's wealth and his/her home region to each transaction, given that both tables contain social security number. As the data about some persons might be missing from the demographic database, miners must agree on predefined constants to use instead of real shares if the encrypted key is missing. In this case, optional post-processing step is needed to hide rows with dummy values. However, the post-processing phase does not hide the number of missing data entries. We discuss this issue in Section 4.3.

Theorem 3. *Let $\mathcal{P} = (\pi_s)$ be a pseudorandom permutation family. If a share-computing framework provides universally composable protocols for database shuffle and oblivious evaluation of $\pi_s(x)$ from secret shared values of x and s, and there are no duplicate key values in any of the input tables, then the* PrpJoin *protocol is universally composable in the computational model.*

Proof (Sketch). For clarity, let us analyse the security in the modified setting where $\mathcal{P}$ is the set of all permutations and Steps 1–4 are performed by trusted third party. Let m be the number of rows in the final database table and $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ the vectors of encrypted values published during PrpJoin protocol. For obvious reasons, $|\boldsymbol{y}_1 \cap \boldsymbol{y}_2| = m$ and the set $\boldsymbol{y}_1 \cup \boldsymbol{y}_2$ consists of $m_1 + m_2 - m$ values, which are chosen randomly from the input domain without replacement. As Step 1 guarantees that the elements in $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are in random order, it is straightforward to simulate $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ given only the number of rows m.

Hence, the simulation of the protocol is straightforward. First, the simulator forwards all input shares and gets back the final output shares and thus learns m. After that it generates shares for the shuffled databases by creating the correct number of valid shares of zero. As the adversarial coalition is small enough, the adversary cannot distinguish them from valid shares. Next, it generates $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ according to the specification given above and forwards the values to the adversary together with properly aligned output shares such that a semihonest adversary would assemble the database of output shares in the correct way.

It is easy to see that the simulation is perfect in the semihonest model. The same is true for the malicious model with honest majority, since honest parties can always carry out all the computations without the help from the adversarial coalition. In case of dishonest majority, the adversarial coalition is allowed to learn its output and then terminate the protocol. In our case, the simulator must terminate the execution when the adversarial coalition decides to stop after learning the encrypted vectors $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$.

We can use the same simulation strategy for the original protocol where the trusted third party uses a pseudorandom permutation family. As the key s is unknown to all parties, the joint output distributions of the real and hybrid worlds are computationally indistinguishable. The latter is sufficient, as security in the hybrid model carries over to the real world through universal composability of share shuffling and oblivious function evaluation protocols. □

Efficiency. By combining the secure oblivious AES evaluation and the oblivious shuffle from [29], we get an efficient instantiation of the PrpJoin protocol. For all database sizes, the resulting protocol does $\Theta(m_1 + m_2)$ share-computing operations and $\Theta(m_1 \log m_1 + m_2 \log m_2)$ public computation operations.[7]

4.2 Secure inner join based on unique multi-column key values

Let us now consider the case when database tables are joined based on several columns, such as name and birth date. We can reduce this kind of secure join to the previous case by using oblivious hashing. An *ε-almost universal hash function* is a function $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ that compresses message into shorter tags so that the following inequality holds:

$$\forall x \neq x' \in \mathcal{M} : \quad \Pr\left[k \leftarrow \mathcal{K} : h(k,x) = h(k,x')\right] \leq \varepsilon .$$

Such a function can be used to reduce the length of the unique key that spans over several columns. However, this function must support efficient oblivious evaluation. The Carter-Wegman construction [16]

$$h(\boldsymbol{k}, \boldsymbol{x}) = x_s k_s + \cdots + x_2 k_2 + x_1 k_1$$

is a good candidate for our application as it consist of a few simple operations and it is $2^{-\ell}$ almost universal when computations are done over the field $\mathbb{F}_{2^\ell}$. Another compelling alternative is to use several independent Carter-Wegman functions over $\mathbb{F}_2$. For ℓ independently chosen keys, the collision probability is still $2^{-\ell}$. In the semihonest model, the communication complexity of the resulting

[7] The theoretical asymptotic complexity is higher, as the size of the database can be only polynomial in the security parameter and thus oblivious PRF evaluation takes $\mathsf{poly}(m)$ steps. Consequently, the protocol is asymptotically more efficient than the naive solution as long as the PRF evaluation is sub-linear in the database size.

Offline phase

1. Generate shared random keys (k_{ij}) for the Carter-Wegman construction.

Online hashing phase

2. Treat each key tuple as a long bit string $\boldsymbol{x} = (x_s, \ldots, x_1)$.
3. Use secure scalar product algorithm to compute the secret shared hash code:

$$h(\boldsymbol{k}_j, \boldsymbol{x}) = x_s k_{sj} + \cdots + x_1 k_{1j} \ .$$

Protocol 2: Oblivious hashing OHASH

oblivious hashing protocols is the same, as the amount of communication scales linearly wrt the bit length. For the malicious models, the trade-offs depend on exact implementation details of multiplication protocol. The resulting algorithm for oblivious hashing is depicted as Protocol 2.

Theorem 4. *If a share-computing framework provides universally composable protocols for addition and multiplication over $\mathbb{F}_2$, the* OHASH *protocol is universally composable in the information theoretical model. For ε-almost universal hash function and m invocations of* OHASH *the probability that two different inputs lead to the same output is upper bounded by $\frac{1}{2}m^2\varepsilon$.*

Proof (Sketch). The claim about security is evident as multiplication together with addition is sufficient to implement scalar product over $\mathbb{F}_2$. The collision probability follows from the union bound $\Pr[\mathsf{collision}] \leq \binom{m}{2} \cdot \varepsilon \leq \frac{m^2\varepsilon}{2}$. $\square$

Efficiency. A collision in the same key column invalidates the assumptions of Theorem 3, whereas a collision between keys of different tables introduces fraudulent entry in the resulting table. Hence, the size of Carter-Wegman construction must be chosen so that the probability of a collision event is negligible. By using 2^{-80} as the failure probability, we get that 128 bit Carter-Wegman construction allows us to operate up to 33.5 million table entries, which is clearly more than a secure database join protocol can handle in feasible time. To handle around million entries with the same failure probability it is sufficient to use 119-bit Carter-Wegman construction. However, note that the standard implementation of OHASH that computes each bit of the MAC separately and thus duplicates the data vector for each bit, has a larger communication complexity than oblivious AES. Experiments show that for 288 bit input and 128-bit output the complexity of a single OHASH is around 25 ms while the amortised complexity is around 5.7 ms. The corresponding numbers are 11 ms and 0.012 ms for the optimised OHASH protocol detailed in Appendix A. To put the results into context, note that unoptimised OHASH is over 10 times slower than oblivious AES, while the optimised OHASH has almost no impact to performance of the equi-join protocol as its running time is around 5%.

4.3 Secure inner join based on non-unique key columns

It is straightforward to see that PrpJoin protocol leaks the number of coinciding keys in the database tables. Without the additional shuffle round at the end of PrpJoin, miners can also find out how many key collisions occur for a particular row in the final database. In certain settings, this might be enough to deduce private information. For instance, the number of purchases might identify some supermarket clients. The extra shuffle step destroys a direct link between database rows and number of collisions. However, collision events are still observable. More formally, let the *occurrence signature* for a key be the number of occurrences in each table. Then we can state the following security claim.

Theorem 5. *If the share-computing framework satisfies the same assumption as in Theorem 3, the* PrpJoin *protocol leaks only the final size of the database and occurrence signatures without the corresponding key values.*

Proof (Sketch). The proof hinges on the fact that the full knowledge of occurrence signatures is sufficient for simulating the outcome of the encryption and join phase and the database shares provided by the trusted third party can be used to correctly align the output of the post-processing phase with the outputs of honest parties. The properties of oblivious shuffle assure that the correspondence between occurrence signatures and keys is not required in simulation. □

Unfortunately, the *occurrence signature* itself might cause privacy breaches. For instance, if we know how many times a person visited a doctor, then we can detect (with certain probability) whether the person is in the database table or not. If the database table contains cancer patients with bad prognosis, the resulting privacy breach may be significant. Similarly, the mere amount of purchase transactions with the same identity might give valuable business insight.

In many cases, the number of rows with the same key value is small, say, less than 10. Moreover, the upper bound ℓ on the number of rows corresponding to the same key might be public, e.g., the maximal number of purchases one can do might be limited. In such cases, we can solve the problem by adding fake rows to the database so that each key occurs exactly ℓ times. The corresponding unification protocol is depicted as Protocol 3.

The mask-and-mix phase can be implemented with standard share-computing operations. The secret-shared binary columns $\boldsymbol{b}$ and $\boldsymbol{c}$ are used to keep track of fake rows. The former is used only inside the protocol while the latter is added to the output table. In the protocol output table the value c_i is 0 for original rows and 1 for added fake rows. After performing the actual join operation on two such tables, we can filter out rows that contain fake entries ($c_i = 1$ for either one or both tables). The oblivious sorting step can be performed in $\Theta(m \log m)$ steps using the AKS sorting network [4] and the ordering predicate[8]

$$(k_i, b_i) \preceq (k_j, b_j) \qquad \Leftrightarrow \qquad k_i \leq k_j \wedge b_i \leq b_j \enspace .$$

[8] For practical database sizes, other networks with $\Theta(m \log^2 m)$ are more efficient but they are still sub-quadratic.

Mask-and-mix phase

1. Add two secret-shared bit columns $\boldsymbol{b}$ and $\boldsymbol{c}$ for marking fake rows.
2. For each row add $\ell - 1$ fake rows with the same key and flags $b_i = 1$ and $c_i = 1$.
3. Apply oblivious shuffle to the database to hide the content.

Sort-and-filtering phase

4. Sort all rows according to the key value and the flag pairs so that fake entries with same keys are always after the non-faked ones.
5. Linearly scan key-flag pairs (k_i, b_i) and set flag b_i to zero if less than ℓ rows with the same key precede the current row.
6. Apply oblivious shuffle to the database and open the index column $\boldsymbol{b}$. Delete all rows, which are still marked as fake, i.e., $b_i = 1$.

Protocol 3: Size unification protocol SUNIF

For the fifth step we need a secret-shared counter n and the update rule:

$$n = \begin{cases} n+1, & \text{if } k_{i-1} = k_i \ , \\ 1, & \text{otherwise} \ , \end{cases} \qquad\qquad b_i = \begin{cases} 0, & \text{if } n \leq \ell \ , \\ 1, & \text{otherwise} \ . \end{cases}$$

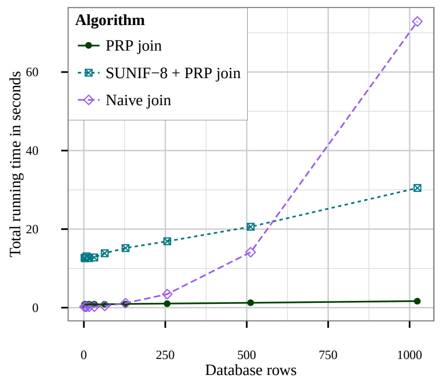
Theorem 6. *Assume that a share-computing framework provides universally composable protocols for arithmetic operations and protocols for oblivious sort and shuffle and the maximal number of keys with the same value is below ℓ. Then the* SUNIF *protocol produces a database where each key appears exactly ℓ times and leaks only the number of distinct keys.*

Proof (Sketch). For the proof note that the third and sixth step destroy all information about the location of fake rows and the number of ones revealed at the last step is determined only by the database size and the number of distinct keys. The simulation construction is analogous to the previous proofs. □

Instead of opening the unused fake rows, we can assign them non-used keys so that all keys occur exactly ℓ times. For instance, we can treat the pair (k_i, b_i) as a new key and use a global counter d for generating new key values in a row. That is, we set $k_i = d$ when $n \geq \ell$ and increment d after ℓ assignments. As a result, we get an new database where each key (k_i, b_i) occurs ℓ times. Let us denote this protocol as SUNIF^{+}.

Corollary 1. *Let the number of key occurrences be bounded by a public constant ℓ. Then by combining* SUNIF^{+} *and* PRPJOIN *protocol we get an equi-join protocol that leaks no information besides the number of rows in the output table.*

Proof (Sketch). The PRPJOIN protocol leaks no information as each key occurs exactly ℓ times. However, the resulting join after the join phase contains rows that consist partly of fake entries. Then, we can use all fake flags c_i to compute whether the row is valid or not in the post processing step and eliminate invalid database entries. Again, this leaks no information as the number of invalid entries is determined by the number of rows in the output table. □



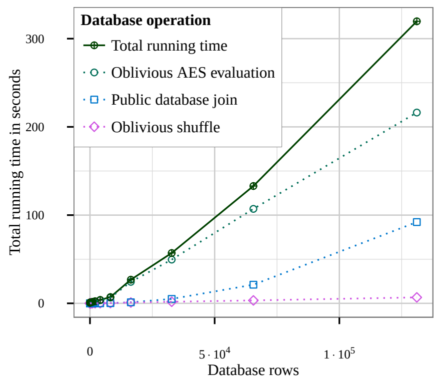


Fig. 2. Benchmarking results for the oblivious database join operation.

Note that the resulting protocol has complexity which is roughly $\Theta(\ell m \log m)$ and thus is much faster than the plain NaiveJoin algorithm with quadratic complexity described in Section 4.

Missing values. The simplest way to handle missing key values is to use a dedicated key value. However, this might increase the maximal number of colliding keys ℓ in the table. To circumvent this, we can add a dedicated unique key for each missing value and add dummy rows with corresponding keys to the other database. Note that this addition does not change the maximal number of colliding keys. Since we do not want to leak which values are missing, we must add an extra row for each row in the first database and the same amount of dummy rows with matching fake keys to the second database. To get rid of extra dummy rows when the key is not missing from the second database, we can use the analogous filtering technique to the Sunif protocol.

4.4 Benchmarking results

We measured the performance of two secure database join protocols with the same setup as we used for timing the oblivious AES evaluation. For the experiment, we measured how much time it takes to join two database tables consisting of five 32-bit columns including the single column key. Both databases were of the same size and each key in one table had exactly one matching key in the other table. For AES, we used the BCirc-1 version of the protocol as it has the lowest amortized cost for tables with thousands of rows.

Results depicted in Figure 2 clearly indicate that PrpJoin protocols is much more efficient even for modest database sizes and it scales nearly linearly. More precisely, the only non-linear performance component is public database join operation, which is known to take $\Theta(m \log m)$ operations. The exact balance between oblivious AES evaluation and database shuffle depends on the number

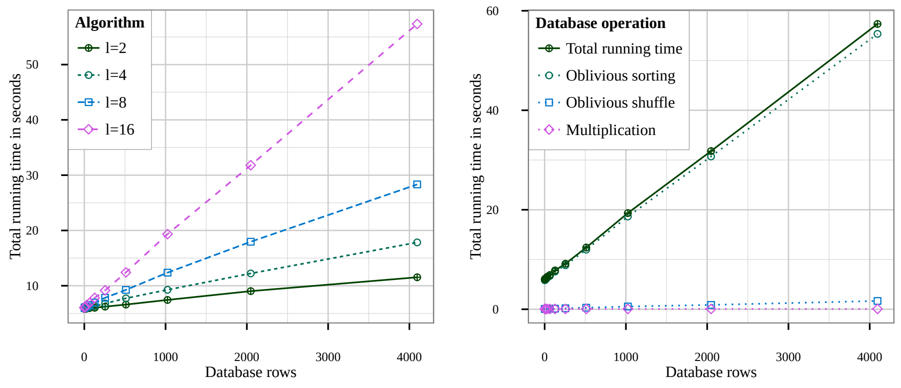


Fig. 3. Benchmarking results for the SUNIF operation.

of columns. As the oblivious database shuffle scales linearly with the number of columns, the fraction of time spent on shuffling increases linearly with the number of columns. However, the slope is rather small.

For instance, consider two database tables with $10,000$ rows each. Then the amount of time spent on oblivious shuffle becomes comparable with oblivious AES evaluation only if the number of columns per table exceeds 180 for our experimental setting. Hence, we can safely conclude that the oblivious database join is feasible in practical applications.

The NAIVEJOIN algorithm spends most of its time doing oblivious database shuffle. The shuffle operation itself is efficient, but the share size of the database is big. Even for two tables consisting of 1000 rows we must shuffle a database with million rows. Hence, it is affordable only for small databases.

To show the tradeoff between both algorithms, Figure 2 contains the running-time of secure database join with SUNIF preprocessing that can handle up to 8 coinciding keys. Although the initial running-time is much worse than NAIVEJOIN due to expensive oblivious sorting step, the algorithm scales near-linearly and quickly becomes faster. The exact tradeoff point depends heavily on the value of ℓ — the larger it is, the bigger the initial slowdown.

For comparison, we also benchmarked the case with non-unique keys. As SUNIF/SUNIF$^+$ protocols are independent of the actual join operation (PRPJOIN) and can be viewed as pre-processing of the database table, we measured the performance of SUNIF separately. As with testing PRPJOIN, the input table of SUNIF has five 32-bit columns, including one key column and varying number of rows. In addition, we have an extra parameter ℓ showing the maximal number of equal keys in the table. To get good scalability, we used oblivious radix sort in the sort-and-filter phase. As the right pane in Figure 3 clearly shows the sorting algorithm indeed scales near-linearly with the data and is thus a good candidate for processing large databases. The figure also shows that there is room for further optimisation – by reducing the complexity of oblivious sorting one can speed-up the algorithm more than ten times before other algorithms start to impact the total working time. The left pane in Figure 3 clearly shows that Sunif scales linearly with the number of rows in database table as expected.

4.5 Comparison with related work

Protocols for privacy-preserving database join have been proposed before. However, none of them are applicable in our model where input and output tables are secret shared. One of the first articles on privacy-preserving datamining showed how exponentiation can be used to compute equi-join in two-party case [3]. However, their protocol reveals the resulting database.

Freedman *et al.* showed how oblivious polynomial evaluation and balanced hashing can be used to implement secure set intersection [20]. The resulting two-party protocol is based on additively homomorphic encryption and has complexity $\Theta(m_1 m_2)$ without balanced hashing. The latter significantly reduces the amount of computations by splitting the elements into small distinct groups. The same idea is not directly applicable in our setting, since our data is secret shared, while their protocol assumes that key columns are local inputs.

Oblivious polynomial evaluation is not very useful in our context, as it is shorthand for the test $x \in \{b_1, \ldots, b_k\}$ which requires $\Theta(k)$ multiplications, while the PrpJoin protocol does all such comparisons publicly.

Hazay and Lindell [23] have also proposed a similar solution that uses pseudorandom permutation to hide initial data values and performs secure set intersection on ciphertexts. However, they are working in a two-party setting where one of the parties learns the intersection.

5 Conclusion

In this paper we showed that there are several compelling ways to implement oblivious AES evaluation in a multi-party setting where the plaintext and the ciphertext are shared between the parties. As the second important contribution, we described and benchmarked efficient protocols for joining secret-shared databases.

Our benchmarking results showed that it is possible to get throughputs around 3500 blocks per second for the oblivious AES, which is the fastest three-party MPC implementation known to the authors. In general, any block cipher based on substitution permutation networks (SPN) is a good candidate for oblivious evaluation as long as the Sbox has low multiplicative complexity and the rest of the cipher is linear over $\mathbb{F}_{2^k}$. Experimental results allow us to conclude that throughput around 350 blocks per second is achievable for any comparable SPN cipher, as the evaluation method of [28] is applicable for any Sbox.

Note that the AES key schedule is appropriate for oblivious evaluation, as all the round keys can be computed on demand. Consequently, the usage of pre-shared round keys reduces the running time for a single operation only by 25%. The only way to get more efficient oblivious evaluation protocols is to use Sbox constructions with smaller multiplicative complexity than 32. However, these Sboxes are also more likely to be weaker against linear cryptanalysis and algebraic attacks. Thus, it would be really difficult to come up with more compelling block cipher for multi-party setting – any secure block cipher designed for the oblivious evaluation, is also a good ordinary block cipher.

For the database join, we showed how to combine oblivious evaluation of almost universal hashing and pseudorandom functions to get a collision resistant pseudorandom function, which can handle arbitrary sized database keys. The resulting PrpJoin protocol works under the assumption that all key column entries are unique. Although we can always fall back to NaiveJoin and preserve security without this restriction, the performance penalty is excessive. A better solution was also proposed in the paper for the case when the number of non-unique occurrences of keys is upper bounded by some predefined quantity ℓ.

From a truly theoretical viewpoint, the question whether sub-quadratic complexity for oblivious database join is achievable depends on existence of pseudorandom functions with low multiplicative complexity. The latter is an interesting open question. Another practically more important open question is to find new almost universal hash function constructions with lower multiplicative complexity or to prove that current constructions are optimal. The circuit complexity of universal hash functions has been studied in the context of energy efficiency [27], the main goal has been minimisation of total circuit complexity which is a considerably different minimisation goal.

Acknowledgments. The work of Riivo Talviste was supported by European Social Fund's Doctoral Studies and Internationalisation Programme DoRa. Also, we would like to thank Helger Lipmaa for insightful theoretical suggestions.

References

1. Raknet – multiplayer game network engine. `http://www.jenkinssoftware.com`.
2. SecureSCM. Technical report D9.1: Secure Computation Models and Frameworks. `http://www.securescm.org`, July 2008.
3. Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 86–97, New York, NY, USA, 2003. ACM.
4. Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in c log n parallel sets. *Combinatorica*, 3(1):1–19, 1983.
5. Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In Kaisa Nyberg, editor, *Proc. of FSE '08*, volume 5086 of *LNCS*, pages 470–488. Springer, 2008.
6. Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS '08: Proceedings of the 15th ACM conference on Computer and Communications Security*, pages 257–266, New York, NY, USA, 2008. ACM.

7. D.J. Bernstein. ChaCha, a variant of Salsa20. `http://cr.yp.to/chacha.html`, 2008.
8. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In Sushil Jajodia and Javier Lopez, editors, *Computer Security – ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer Berlin / Heidelberg, 2008.
9. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. Cryptology ePrint Archive, Report 2011/662, 2011. `http://eprint.iacr.org/`.
10. Peter Bogetoft, Dan Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Nielsen, Jesper Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, volume 5628 of *LNCS*, pages 325–343. Springer Berlin / Heidelberg, 2009.
11. Joan Boyar and René Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *LNCS*, pages 178–189. Springer Berlin / Heidelberg, 2010.
12. Joan Boyar and René Peralta. A small depth-16 circuit for the aes s-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *SEC*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.
13. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the USENIX Security Symposium '10*, pages 223–239, Washington, DC, USA, 2010.
14. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
15. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: 42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, 2001.
16. Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
17. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.
18. Ivan Damgård and Marcel Keller. Secure multiparty aes. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 367–374. Springer, 2010.
19. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [34], pages 643–662.
20. Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer ScienceLNCS*, pages 1–19. Springer Berlin / Heidelberg, 2004.
21. Craig Gentry and Shai Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. In *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 129–148. Springer, 2011.
22. Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.

23. Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standardsmartcards. In *ACM Conference on Computer and Communications Security*, pages 491–500, 2008.
24. Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *CCS '10: Proceedings of the 17th ACM conference on Computer and Communications Security*, pages 451–462. ACM, 2010.
25. Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *20th USENIX Security Symposium*, pages 8–12, 2011.
26. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.
27. Jens-Peter Kaps, Kaan Yuksel, and Berk Sunar. Energy scalable universal hashing. *IEEE Trans. Comput.*, 54(12):1484–1495, December 2005.
28. John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In Peter Thiemann and Robby Bruce Findler, editors, *ICFP*, pages 189–200. ACM, 2012.
29. Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In Xuejia Lai, Jianying Zhou, and Hui Li, editors, *Information Security*, volume 7001 of *LNCS*, pages 262–277. Springer Berlin / Heidelberg, 2011.
30. Lior Malka. Vmcrypt: modular software architecture for scalable secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 715–724, New York, NY, USA, 2011. ACM.
31. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). *Federal Information Processing Standards Publications*, FIPS-197, 2001.
32. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Safavi-Naini and Canetti [34], pages 681–700.
33. Benny Pinkas, Thomas Schneider, Nigel Smart, and Stephen Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer Berlin / Heidelberg, 2009.
34. Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

A Carter-Wegman MAC protocol proof

As the computation of Carter-Wegman hash function is essentially a matrix-vector multiplication over the field $\mathbb{F}_2$, we can use an optimisation technique, which is applicable in many other matrix multiplication settings. The corresponding protocol is depicted as Protocol 4. We use double brackets to denote secret shared values, e.g. the secret shared version of $s = s_1 \oplus s_2 \oplus s_3$ is shown as $[\![s]\!]$, where party $\mathcal{P}_i$ holds s_i. For double indices, the second index shows which

Input-oputput specification

Protocol input is a shared s-bit value $[\![m]\!]$ and shared s-bit keys $[\![k_1]\!], \ldots, [\![k_\ell]\!]$.
Protocol output is a shared ℓ-bit MAC value $[\![c]\!]$.

Precomputation phase

1. Each miner $\mathcal{P}_i$ generates ℓ random bits $r_i^1, \ldots, r_i^\ell \leftarrow \mathbb{Z}_2$.

Data distribution phase

3. Miner $\mathcal{P}_1$ sends s-bit shares $m_1, k_{1,1}, \ldots, k_{\ell,1}$ to $\mathcal{P}_2$.
   Miner $\mathcal{P}_2$ sends s-bit shares $m_2, k_{1,2}, \ldots, k_{\ell,2}$ to $\mathcal{P}_3$.
   Miner $\mathcal{P}_3$ sends s-bit shares $m_3, k_{1,3}, \ldots, k_{\ell,3}$ to $\mathcal{P}_1$.

Post-processing phase

5. Each miner $\mathcal{P}_i$ computes $w_{ij}^t \leftarrow m_i^t \wedge k_{j,i}^t \oplus m_{i-1}^t \wedge k_{j,i}^t \oplus m_i^t \wedge k_{j,i-1}^t$
   for each key $j \in \{1, \ldots, \ell\}$ and bit $t \in \{1, \ldots, s\}$ and sums them
   up together with re-randomisation $c_i^j \leftarrow w_{ij}^1 \oplus \cdots \oplus w_{ij}^s \oplus \oplus r_i^j \oplus r_{i-1}^j$.

Protocol 4: More efficient protocol for Carter-Wegman MAC

party holds the bitstring and the first shows for which output bit it will be used for. Since all values are bitwise shared, we can operate with individual bits of the shares. Operations on individual bits use superscript bit index notation.

Theorem 7. *Assume that the shares of m are correctly generated. Then Protocol 4 is correct and secure against single passively corrupted miner.*

Proof (Sketch). For each bit c^j of MAC the correctness follows from

$$\begin{aligned}[\![c^j]\!] &= \bigoplus_{i=1}^{3} \left(\bigoplus_{t=1}^{s} m_i^t \wedge k_{j,i}^t \oplus m_{i-1}^t \wedge k_{j,i}^t \oplus m_i^t \wedge k_{j,i-1}^t \right) \oplus r_i^j \oplus r_{i-1}^j \\ &= \bigoplus_{i=1}^{3} \bigoplus_{t=1}^{s} \left(m_i^t \wedge k_{j,i}^t \oplus m_{i-1}^t \wedge k_{j,i}^t \oplus m_i^t \wedge k_{j,i-1}^t \right) = \bigoplus_{t=1}^{s} (m^t \wedge k_j^t) = h(k_j, m)\end{aligned}$$

since the inner most sum contains all combinations of $m_a \wedge k_b$.

For the security analysis, it is sufficient to consider the corruption of $\mathcal{P}_2$ who receivers all shares owned by $\mathcal{P}_1$. Note that two shares out of three have always uniform distribution. Hence, it is trivial to simulate all messages received by $\mathcal{P}_2$. Since $\mathcal{P}_2$ is semihonest, the simulator can extract shares of the message and keys from the input of $\mathcal{P}_2$ and submit them to the trusted party who will return shares $c_2^1, \ldots, c_2^\ell$. Since the simulator knows what random values $r_2^1, \ldots, r_2^\ell$ $\mathcal{P}_2$ is going to use, it can pick $r_1^1, \ldots, r_1^\ell$ so that $\mathcal{P}_2$ will indeed output $c_2^1, \ldots, c_2^\ell$. We leave the detailed analysis of the simulation construction to the reader. □