SCISPACE
formerly Typeset

Open access • Book Chapter • DOI:10.1007/978-3-642-39884-1_23

GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits

— Source link

Thomas Schneider, Michael Zohner

Institutions: Technische Universität Darmstadt

Published on: 01 Apr 2013 - Financial Cryptography

Topics: Secure two-party computation

Related papers:

- Improved Garbled Circuit: Free XOR Gates and Applications
- How to play ANY mental game
- How to generate and exchange secrets
- Extending Oblivious Transfers Efficiently
- Faster secure two-party computation using garbled circuits

Share this paper:

View more about this paper here: https://typeset.io/papers/gmw-vs-yao-efficient-secure-two-party-computation-with-low-4v6w7wz2oi

GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits

Thomas Schneider and Michael Zohner

Engineering Cryptographic Protocols Group (ENCRYPTO),
European Center for Security and Privacy by Design (EC SPRIDE),
Technische Universität Darmstadt, Germany
{thomas.schneider,michael.zohner}@ec-spride.de

Abstract. Secure two-party computation is a rapidly emerging field of research and enables a large variety of privacy-preserving applications such as mobile social networks or biometric identification. In the late eighties, two different approaches were proposed: Yao's garbled circuits and the protocol of Goldreich-Micali-Wigderson (GMW). Since then, research has mostly focused on Yao's garbled circuits as they were believed to yield better efficiency due to their constant round complexity.

In this work we give several optimizations for an efficient implementation of the GMW protocol. We show that for semi-honest adversaries the optimized GMW protocol can outperform today's most efficient implementations of Yao's garbled circuits, but highly depends on a low network latency. As a first step to overcome these latency issues, we summarize depth-optimized circuit constructions for various standard tasks. As application scenario we consider privacy-preserving face recognition and show that our optimized framework is up to 100 times faster than previous works even in settings with high network latency.

Keywords: GMW protocol, optimizations, privacy-preserving face recognition.

1 Introduction

Generic secure two-party computation allows two parties to jointly compute any function on their private inputs without revealing anything but the result. Interestingly, two different approaches have been introduced at about the same time in the late eighties: Yao's garbled circuits [33] and the protocol of Goldreich-Micali-Wigderson (GMW) [11, 12]. Both protocols allow secure evaluation of a function that is represented as a Boolean circuit and, in their basic form, provide security against semi-honest adversaries who honestly follow the protocol but try to learn additional information from the observed messages – this widely used model allows to construct highly efficient protocols and is the focus of our paper.

As Yao's protocol has a constant number of rounds and requires *Oblivious Transfers (OTs)* only for the inputs of one of the parties while the secure evaluation of a gate requires only symmetric cryptographic operations, it was believed

A.-R. Sadeghi (Ed.): FC 2013, LNCS 7859, pp. 275–292, 2013.
© Springer-Verlag Berlin Heidelberg 2013

to be more efficient than the GMW protocol, which requires an interactive OT for each AND gate (see for example [13, Sect. 1.2]). In fact, many subsequent works presented improvements to Yao's protocol and showed that it can be made truly practical and applied to a large variety of privacy-preserving applications, e.g., [14, 15, 19, 20, 24, 29].

In a recent work [5], it was shown that by implementing OTs efficiently using symmetric cryptographic primitives, the semi-honest version of the GMW protocol can outperform previous secure computation protocols with $n \geq 3$ parties. However, for the two-party case, the authors of [5] state that they expect their implementation to be roughly a factor of two slower than today's fastest implementation of Yao's garbled circuit protocol of [15]. For stronger active adversaries, it has been shown in [27] that an extension of the GMW protocol achieves a performance that can compete with garbled circuit-based protocols.

1.1 Outline and Our Contributions

In this work we show that the GMW protocol is truly practical in the setting with two semi-honest parties and in fact has some advantages over Yao's garbled circuits protocol. Our contribution is threefold:

Depth-Efficient Circuits. As the GMW protocol requires interaction for the secure evaluation of AND gates, the dependence on the latency can be reduced by using circuit constructions that have both, low size and depth. We give a summary of such circuit constructions for standard functionalities in §3.

Implementation-Specific Optimizations. We extend the implementation of the GMW protocol of [5] with various optimizations to yield better performance in the setting with two parties. Our optimizations are described in §4 and include load balancing (to distribute the workload equally among the parties) and processing multiple gates in parallel. Overall, our optimized implementation yields a more than 10 times faster runtime compared to the implementation of [5] for an example circuit with about 800,000 gates.

Performance Evaluation. Finally, in §5, we compare the optimized GMW protocol with today's most efficient techniques for garbled circuits on a conceptual level (independent of the implementation). Afterwards, we experimentally measure and evaluate the performance of our implementation for two example applications: a mobile social network [5] and privacy-preserving face recognition using Eigenfaces [9, 14, 30] and the Hamming distance [28] in a desktop environment that has a network with high bandwidth. In settings with low network latency our implementation is able to process up to 1,000,000 AND gates (1-out-of-4 OTs) per second in the setup phase and 50,000,000 AND gates per second in the online phase. Our evaluation indicates that the GMW protocol is a noticeable alternative to previous protocols based on garbled circuits.

1.2 Related Work

Circuit Optimizations. Since the communication complexity of garbled circuits is independent of the depth of the evaluated circuit, cryptographic literature

mostly focused on minimizing the multiplicative size of circuits, i.e., the number of AND gates where XOR gates are free, e.g. [4,18,29]. Most of the existing work on minimizing circuit depth originates from the area of electrical engineering (cf. [8,31,32]), where the circuit depth directly affects the computation latency. Our setting for circuit optimization is slightly different from this as we can evaluate XOR gates for free, can store intermediate results indefinitely, and have unbounded fan-out.

Garbled Circuit Frameworks. Starting with Fairplay [24] that demonstrated the practical feasibility of Yao's garbled circuits, various frameworks for secure two-party computation have been developed. The currently fastest garbled circuits framework with security in the semi-honest model is [15]. A garbled circuit framework secure against malicious adversaries was given in [20].

GMW Frameworks. The implementation of [5] showed that for semi-honest adversaries the GMW protocol can outperform previous secure multi-party computation frameworks for circuits that can be efficiently represented as Boolean circuits. However, they show this only for the setting with $n \geq 3$ parties and expect that their implementation requires about twice the computation time of [15] in the two-party setting (cf. [5, Sect. 1.2]). The GMW protocol was recently extended into a new approach for secure two-party computation with security against malicious adversaries in [27].

Other Frameworks. For completeness we mention that there exist also other approaches for secure two-party computation (beyond garbled circuits and the GMW protocol) that evaluate arithmetic instead of Boolean circuits. These approaches use additively homomorphic encryption as implemented in the VIFF framework [6] or the SPDZ framework [7] and can even be combined with garbled circuits as implemented in [14]. Although these approaches are well-suited for outsourcing computations in scenarios where one party has substantially more computing power than the other party (e.g., cloud computing), they involve relatively expensive public-key operations in the online phase. Hence, these works are orthogonal to the protocols we consider that require only fast operations per gate and allow to distribute the workload equally among the parties.

2 Preliminaries

2.1 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic protocol executed between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$ in which $\mathcal{R}$ obliviously selects one of the inputs provided by $\mathcal{S}$. More specifically, in 1-out-of-n OT_ℓ^m, $\mathcal{S}$ provides m n-tuples $(x_{11}, \ldots, x_{1n}), \ldots, (x_{m1}, \ldots, x_{mn})$ of ℓ-bit strings; $\mathcal{R}$ provides m selection numbers $r_1, \ldots, r_m$ with $1 \leq r_i \leq n$ and obtains x_{jr_j} $(1 \leq j \leq m)$ as output. The widely used Naor-Pinkas OT protocol [25] is secure against semi-honest adversaries under the Decisional Diffie-Hellman (DDH) assumption in the random-oracle model and requires both parties to perform $\mathcal{O}(m)$ modular exponentiations. The following two techniques can be used to substantially speed up OTs.

OT pre-computations [2] allows to pre-compute the OTs on random inputs and later on in the online phase use these pre-computed values as one-time pads to run the OT on the actual inputs. In the online phase, $\mathcal{R}$ sends one message of size $m \log_2 n$ bits to $\mathcal{S}$ who sends back a message of size mnl bits.

OT extensions [16,22] allow to perform a large number of m OTs (OT_ℓ^m) using a small number of t base OTs on t-bit keys (OT_t^t), where t is a security parameter. The marginal cost for each additional OT is a small number of evaluations of a cryptographic hash function (modeled as random oracle) and of a pseudo-random function. More specifically, for each of the m OTs, $\mathcal{S}$ computes n hash evaluations and $t(1 + \log_2 n)$ pseudo-random bits, whereas $\mathcal{R}$ computes 1 hash evaluation and $t(1 + n)$ pseudo-random bits. The communication complexity is one message from $\mathcal{R}$ to $\mathcal{S}$ of size mnt bits and one in the opposite direction of size $mn\ell$ bits.

2.2 Approaches for Secure Two-Party Computation

We summarize the main approaches for secure two-party computation of Boolean circuits next: Yao's garbled circuits (§2.2.1) and the GMW protocol (§2.2.2).

2.2.1 Yao's Garbled Circuits Protocol [33]. The basic idea of Yao's garbled circuits is to let one party, called creator, encrypt the function to be computed. For this, the plain values are mapped to random-looking symmetric keys and for each gate an encryption table is generated that allows to compute the gate's output key given its input keys. The creator then transmits the encrypted circuit along with the corresponding encrypted inputs to the other party, called evaluator. The creator sends his encrypted inputs directly to the evaluator and the evaluator obtains his encrypted inputs obliviously via 1-out-of-2 OT. The evaluator then uses the encrypted inputs to evaluate the encrypted function gate by gate. Finally, the creator provides a mapping from the encrypted output to plain output. As the evaluation of Yao's garbled circuits is performed non-interactively, the resulting protocol has a constant number of rounds.

The following extensions enhance the speed of Yao's garbled circuits protocol: point-and-permute [24], free XOR [19], efficient encryption with a cryptographic hash function [23], garbled row reduction [26, 29], and pipelining [15]. Put together, these techniques allow "free" evaluation of XOR gates (i.e., no communication and negligible computation), interweaving circuit generation and evaluation, and per non-XOR gate 4 evaluations of a cryptographic hash function for the creator, transmissions of an encrypted gate table with 3 entries, and 1 evaluation of a cryptographic hash function for the evaluator.

2.2.2 GMW Protocol [11, 12]. In the GMW protocol two parties interactively compute a function using secret-shared values. For this, the value v of each input and intermediate wire is shared among the two parties with a 2-out-of-2 secret sharing scheme such that each party holds a random-looking share v_i with $v = v_1 \oplus v_2$. As XOR is an associative operation, XOR gates can be securely evaluated locally by XORing the shares. For secure evaluation of AND gates,

the parties run an interactive protocol using one of the two techniques described below. We note that AND gates of the same layer in the circuit can be computed in parallel. Finally, the parties send the respective shares of the output wires to the party that should obtain the output.

Oblivious Transfers. To securely evaluate an AND gate on input shares x_1, x_2 and y_1, y_2, the two parties can run a 1-out-of-4 OT_1^1 protocol. Here, the chooser inputs its shares x_1, y_1 and the sender chooses a random output share z_2 and provides four inputs to the OT protocol such that the chooser obliviously obtains its output share $z_1 = z_2 \oplus ((x_1 \oplus x_2) \wedge (y_1 \oplus y_2))$. As described in §2.1, all OTs can be moved into a pre-processing phase such that the online phase is highly efficient (only two messages and inexpensive one-time-pad operations).

Multiplication Triples. An alternative method to securely evaluate an AND gate on input shares x_1, x_2 and y_1, y_2 are *multiplication triples* [1]. Multiplication triples are random shares a_i, b_i, c_i satisfying $(c_1 \oplus c_2) = (a_1 \oplus a_2) \wedge (b_1 \oplus b_2)$. They can be generated in the setup phase using a 1-out-of-4 OT_1^1 protocol in a similar way to the OT-based solution described above. In the online phase the parties use these pre-generated multiplication triples to mask the input shares of the AND gate, exchange $d_i = x_i \oplus a_i$ and $e_i = y_i \oplus b_i$, and compute $d = d_1 \oplus d_2$ and $e = e_1 \oplus e_2$. The output shares are computed as $z_1 = (d \wedge e) \oplus (b_1 \wedge d) \oplus (a_1 \wedge e) \oplus c_1$ and $z_2 = (b_2 \wedge d) \oplus (a_2 \wedge e) \oplus c_2$. The advantage of multiplication triples over the OT-based solution is that, per AND gate, each party needs to send only one message (independent of each other) and the size of the messages is slightly smaller ($2 + 2$ bits instead of $2 + 4$ bits).

2.3 Evaluation Metrics and Notation

Motivated by the fact that both approaches for secure two-party computation summarized in §2.2 provide free XORs, we consider the *(multiplicative) size* $\mathbf{S}(\text{C})$ of a circuit C as the number of AND gates in C and the *(multiplicative) depth* $\mathbf{D}(\text{C})$ as the maximum number of AND gates on any path from an input to an output of C.

We denote a bit sequence of length ℓ bits as x^ℓ and use x_i to refer to its i-th bit, starting with the least-significant bit x_1. $\bar{x}^\ell$ denotes the bitwise complement of x^ℓ. $x^\ell_{1...n}$ is a sequence of n ℓ-bit values $x_1^\ell, x_2^\ell, ..., x_n^\ell$. A circuit C that processes n values of $(\ell_1, \ell_2, ..., \ell_n)$-bits each is denoted by $\text{C}^{((\ell_1,\ell_2,...,\ell_n),n)}$. If all inputs n have the same length ℓ, we shorten the notation to $\text{C}^{(\ell,n)}$. When n is clear from the context we use C^ℓ instead. We use the standard notations for binary operations, i.e. concatenation $||$, bitwise XOR $\oplus$, bitwise AND $\wedge$, and bitwise OR $\vee$.

3 Circuit Constructions with low Depth and Size

In the following we briefly summarize circuit building blocks that have low depth and only a slightly larger size compared to constructions that are optimized for size only. A detailed list of constructions is given in Appendix A, Tab. 7 (due to space restrictions we defer details to the full version).

3.1 Addition

The standard method for adding two ℓ-bit numbers is the *Ripple-carry* adder ADD_{RC}^{ℓ} that has linear size and depth, $\mathbf{S}(\text{ADD}_{RC}^{\ell})$=$\mathbf{D}(\text{ADD}_{RC}^{\ell}) = \ell$ [18]. In the following we summarize techniques for addition in sub-linear depth.

3.1.1 Ladner-Fischer Adder. The *Ladner-Fischer* adder ADD_{LF}^{ℓ} [21] is a so-called parallel prefix adder that adds two ℓ-bit values x^{ℓ} and y^{ℓ} in logarithmic depth. The idea of parallel prefix adders is to evaluate multiple carry-bits in parallel. During the computation of the sum, the Ladner-Fischer adder computes a parity bit $p_{i,j}$ and a carry bit $c_{i,j}$ in each node at bit position i ($1 \leq i \leq \ell$) and level j ($0 \leq j \leq \lceil \log_2 \ell \rceil$). At level 0, the Ladner-Fischer adder computes $p_{i,0} = x_i \oplus y_i$ and $c_{i,0} = x_i \wedge y_i$. Then, for every node at level $j > 0$, the parity and carry bit are computed as $p_{i,j} = p_{i,j-1} \wedge p_{k,j-1}$ and $c_{i,j} = (p_{i,j-1} \wedge c_{k,j-1}) \vee c_{i,j}$, where k is the node that propagates the carry-bit to position i. Lastly, at level $\lceil \log_2 \ell \rceil + 1$, the sum $s^{\ell+1}$ is computed as $s_{\ell+1} = c_{\ell,\lceil \log_2 \ell \rceil}$, $s_i = p_{i,0} \oplus c_{i-1,\lceil \log_2 (i-1) \rceil}$ for $1 < i \leq \ell$, and $s_1 = p_{1,0}$. The Ladner-Fischer adder has size $\mathbf{S}(\text{ADD}_{LF}^{\ell}) = 1.25\ell\lceil \log_2 \ell \rceil + \ell$ and depth $\mathbf{D}(\text{ADD}_{LF}^{\ell}) = 2\lceil \log_2 \ell \rceil + 1$.

3.1.2 Carry-Save Adder. The *carry-save* adder $\text{ADD}_{CSA}^{(\ell,3)}$ [8] converts the sum of three ℓ-bit unsigned integers x^{ℓ}, y^{ℓ}, and z^{ℓ} into two $\ell+1$-bit unsigned integers $p^{\ell+1}$ and $c^{\ell+1}$ such that $p^{\ell+1} + c^{\ell+1} = x^{\ell} + y^{\ell} + z^{\ell}$. To obtain the result of the addition in sub-linear depth, $p^{\ell+1}$ and $c^{\ell+1}$ can again be added using the Ladner-Fischer adder (cf. §3.1.1). The carry-save adder is composed from ℓ 1-bit full-adders that compute the parity $p_i = x_i \oplus y_i \oplus z_i$ and the carry $c_{i+1} = z_i \oplus ((z_i \oplus x_i) \wedge (z_i \oplus y_i))$ in parallel for every bit position $1 \leq i \leq \ell$. Finally, $p_{\ell+1}$ and c_1 are set to 0.

The carry-save adder has linear size and constant depth, allowing three ℓ-bit numbers to be added with a circuit of size $\mathbf{S}(\text{ADD}_{CSA}^{(\ell,3)}) = \ell + \mathbf{S}(\text{ADD}_{LF}^{(\ell+1)})$ and depth $\mathbf{D}(\text{ADD}_{CSA}^{(\ell,3)}) = 1 + \mathbf{D}(\text{ADD}_{LF}^{(\ell+1)})$. Multiple carry-save adders can be combined to create a carry-save network $\text{ADD}_{CSN}^{(\ell,n)}$ that converts n ℓ-bit numbers to two $\ell + \lceil \log_2 n \rceil$-bit numbers and adds them using $\text{ADD}_{LF}^{\ell+\lceil \log_2 n \rceil}$ with $\mathbf{S}(\text{ADD}_{CSN}^{(\ell,n)}) = (\ell-1)(n-2) + \sum_{i=0}^{\lceil \log_2 n \rceil} i\frac{n}{2^i} + \mathbf{S}(\text{ADD}_{LF}^{(\ell+\lceil \log_2 n \rceil)}) \approx \ell n - 2\ell + n - \lceil \log_2 n \rceil + \mathbf{S}(\text{ADD}_{LF}^{(\ell+\lceil \log_2 n \rceil)})$ and $\mathbf{D}(\text{ADD}_{CSN}^{(\ell,n)}) = \lceil \log_2 n \rceil + \mathbf{D}(\text{ADD}_{LF}^{(\ell+\lceil \log_2 n \rceil)})$ [31].

3.2 Squaring

Although the square of a number can be computed with a multiplication circuit, a squaring circuit is smaller by a factor of about two. The standard school method multiplication circuit MUL^{ℓ} [18] computes the product $x^{\ell}x^{\ell}$ as $\sum_{i=1}^{\ell} 2^{i-1}(x^{\ell}x_i)$. Since each $x_j x_i$ with $i \neq j$ is computed twice, $x_j x_i + x_i x_j$ can be simplified to $2x_i x_j$ and $x_i x_i$ can be replaced with x_i [34]. The corresponding depth-efficient squarer circuit SQR_{LF}^{ℓ} has size $\mathbf{S}(\text{SQR}_{LF}^{\ell}) = \ell^2 + 1.25\ell\lceil \log_2 \ell \rceil - 1.5\ell - 2$ and depth $\mathbf{D}(\text{SQR}_{LF}^{\ell})$=$\mathbf{D}(\text{ADD}_{CSA}^{(2\ell,\lceil \ell/2 \rceil)})$+$\mathbf{D}(\text{ADD}_{LF}^{2\ell})$+1 = $3\lceil \log_2 \ell \rceil + 3$.

3.3 Comparison

A circuit that checks the equality of two values EQ^ℓ has linear size $\mathbf{S}(\mathrm{EQ}^\ell) = \ell - 1$ [19] and can be built in a pairwise tournament fashion to achieve logarithmic depth $\mathbf{D}(\mathrm{EQ}^\ell) = \lceil \log_2 \ell \rceil$. The standard greater than circuit GT^ℓ_S that checks whether one number is greater than another has linear size, but also linear depth $\mathbf{S}(\mathrm{GT}^\ell_S) = \mathbf{D}(\mathrm{GT}^\ell_S) = \ell$ [18]. The greater than operation can be computed in logarithmic depth using the circuit GT^ℓ_{DC} of [10] that recursively splits the ℓ-bit parameters into half. More precisely, let $x^\ell = (x_H || x_L)$ and $y^\ell = (y_H || y_L)$ be two ℓ-bit integers with x_H, y_H being $\lceil \frac{\ell}{2} \rceil$-bit and x_L, y_L being $\lfloor \frac{\ell}{2} \rfloor$-bit unsigned integers. GT^ℓ_{DC} is then recursively computed as $\mathrm{GT}^\ell_{DC}(x^\ell, y^\ell) = \mathrm{GT}^{\lceil \frac{\ell}{2} \rceil}_{DC}(x_H, y_H) \oplus \mathrm{EQ}^{\lceil \frac{\ell}{2} \rceil}(x_H, y_H) \wedge \mathrm{GT}^{\lfloor \frac{\ell}{2} \rfloor}_{DC}(x_L, y_L)$ until x and y are bits for which $\mathrm{GT}^1_{DC}(x_i, y_i) = x_i \wedge \bar{y}_i$. The circuit has size $\mathbf{S}(\mathrm{GT}^\ell_{DC}) = 3\ell - \lceil \log_2 \ell \rceil - 2$ and depth $\mathbf{D}(\mathrm{GT}^\ell_{DC}) = \lceil \log_2 \ell \rceil + 1$.

3.4 Hamming Weight

The Hamming weight circuit CNT^ℓ counts the number of one entries in x^ℓ, i.e., it computes its Hamming weight $d_H(x^\ell) = \sum_{i=1}^{\ell} x_i$. In [4], a Hamming weight circuit CNT^ℓ_{BP} was given that splits a value x^ℓ into three parts of length $m = \lceil \frac{\ell - 1}{2} \rceil$, $n = \lfloor \frac{\ell - 1}{2} \rfloor$, and one bit, respectively: $x^\ell = (x^m || x^n || x_1)$. CNT^ℓ_{BP} is then computed recursively as $\mathrm{CNT}^\ell_{BP}(x^\ell) = \mathrm{ADD}^{\lceil \log_2 \ell \rceil}_{RC}(\mathrm{CNT}^m_{BP}(x^m), \mathrm{CNT}^n_{BP}(x^n), x_1)$ where x_1 can be provided as carry-in to the addition circuit at no extra cost. The circuit has size $\mathbf{S}(\mathrm{CNT}^\ell_{BP}) = \ell - d_H(\ell)$ and depth $\mathbf{D}(\mathrm{CNT}^\ell_{BP}) = \lfloor \log_2 \ell \rfloor$.

4 Optimizations for Two-Party GMW

The original paper of [5] provides an implementation of the GMW protocol and gives performance numbers for $n \geq 3$ parties. For two parties, they expect that their implementation is slower than the currently fastest garbled circuit implementation of [15] by a factor of two (cf. [5, Sect. 1.2]). We modified and extended their GMW implementation for better efficiency in the two-party setting. In the following we give an overview of our modifications that improved the overall performance the most. We list the modifications in the order they were implemented and summarize the performance numbers in Tab. 1. Hence, the performance numbers for each modification include the improvement of all previous optimizations. In total, we improve the overall runtime of the following example application by more than a factor of 10.

Benchmarking Environment. In our experiments we evaluate the time for the *setup phase* (Naor-Pinkas OTs and OT extension), *online phase* (sharing of inputs, circuit evaluation, and combining output shares), and the *overall time* (circuit construction, setup phase, and online phase). We perform the comparison on an unoptimized 512-bit multiplication circuit C with $\mathbf{S}(\mathrm{C}) = 800{,}227$ and

$\mathbf{D}(\mathrm{C}) = 38$ using the average time of 100 executions. For the Naor-Pinkas OT we use the group $\mathbb{Z}_p^*$ with $|p| = 512$ bit, also used in [5]. For the OT extensions we set the security parameter to $t = 80$. The server and client run on two 2.5 GHz Intel Core2Quad CPU (Q8300) Desktop PCs with 4 GB RAM each that are connected via Gigabit LAN with a ping latency of 0.2 ms.

Table 1. Time improvements for the individual optimizations

Optimization	None [5]	MT §4.1	PRF §4.2	LB §4.3	GMP §4.4	Bytewise §4.4	SHA-1 §4.4	SIMD §4.5
Setup Phase [s]	13.39	13.82	11.41	7.41	6.87	1.68	0.89	0.84
Online Phase [s]	0.73	0.70	0.70	0.71	0.70	0.31	0.32	0.012
Overall Time [s]	14.19	14.52	12.16	8.14	7.60	2.10	1.35	0.85

4.1 Multiplication Triples

To reduce the impact of the latency during the online time we use Beaver's multiplication triples [1] instead of pre-computed OTs for secure evaluation of AND gates (cf. §2.2.2). This results in a slightly slower setup phase (13.82 s instead of 13.39 s) due to the additional overhead for computing c_2 (instead of using random inputs). However, the online phase gets more efficient (0.70 s instead of 0.73 s) as we only require one instead of two interaction steps and 4 instead of 6 bits sent per AND gate.

4.2 Using AES Instead of SHA as Pseudo-Random Function

The original implementation of [5] used SHA-1 not only for instantiating the random oracle, but also for generating pseudo-random values in the OT extension. To reduce the computational complexity, we use AES in the counter mode as pseudo-random function (PRF). More precisely, we benchmarked the SHA-1 implementation of PolarSSL that was used in [5] against the SHA-1 implementation (sha1-x86_64) and the AES128 (aes-x86_64) implementation of OpenSSL v. 1.0.1c. The results for 10^9 iterations are depicted in Tab. 2. This decreased the number of expensive hash function calls per AND gate (to instantiate the random oracle) from 3.5 to 1 for the receiver $\mathcal{R}$ and from 4.5 to 4 for the sender $\mathcal{S}$; additionally, $\mathcal{R}$ performs 3.1 AES calls and $\mathcal{S}$ performs 0.65 AES calls per AND gate (to instantiate the PRF). Overall, using AES as PRF decreased the setup time from 13.82 s to 11.41 s. We note that the performance could be improved even further by using the Intel AES New Instructions (AES-NI) provided by recent CPUs.

4.3 Load Balancing

In the OT extension protocol executed in the setup phase, the sender and the receiver have different computational workload (cf. §4.2). As the multiplication

Table 2. Comparison of SHA-1 and AES128 implementations for 10^9 iterations

Algorithm	Iterations/s	Bits/s
SHA-1 PolarSSL	$1.30 \cdot 10^6$	$2.08 \cdot 10^8$
SHA-1 OpenSSL	$3.65 \cdot 10^6$	$5.83 \cdot 10^8$
AES128 OpenSSL	$4.99 \cdot 10^6$	$6.39 \cdot 10^8$

triples used in the online phase (cf. §4.1) are symmetric, we can run the OTs to generate them in the setup phase in either direction. Hence, to balance the workload, we run two instantiations of the OT protocol (each for half of the AND gates) in parallel with the roles reversed. With this optimization, each party has the same workload: 2.5 SHA-1 invocations and 1.8 AES invocations per AND gate. Note that now we also need to run the Naor-Pinkas OT protocol for the seed OTs twice, which however amortizes fairly quickly (35 ms computation time and 10 kByte to be transferred). Since the original implementation of [5] already used multi-threading for implementing the OT extensions, we now use four parallel threads during the setup phase (two for each role such that one thread evaluates the pseudo-random function in the first round and the other the random oracle in the second round of the protocol). Overall, load balancing decreased the setup time from 11.41 s to 7.41 s.

4.4 Implementation-Specific Optimizations

In order to further speed up the execution time we performed several implementation-specific optimizations as summarized next.

Arithmetic. For modular arithmetics within the Naor-Pinkas base OTs we replaced the Number Theory Library (*NTL*) v. 5.5.2 used in [5] with the GNU Multiple Precision Arithmetic Library (*GMP*) v. 5.0.5. This decreased the time for the Naor-Pinkas base OTs from 590 ms using NTL to 35 ms using GMP for modular operations on 512 bit values.

Bytewise Operations. A major bottleneck was the bitwise processing order during the OT extension step and the online phase. We reduced the impact of the processing order by performing the operations bytewise instead of bitwise. For the setup phase we thereby gained a decrease in time from 6.87 s to 1.68 s. For the online phase we achieved a decrease in time from 0.70 s to 0.31 s.

SHA-1. Afterwards, the evaluation of SHA-1 for the random oracle became the major bottleneck for the OT extension. We replaced the implementation of SHA-1 from PolarSSL used in [5] with an assembler implementation of OpenSSL v. 1.0.1c (sha1-x86_64). This decreased the setup time from 1.68 s to 0.89 s.

4.5 Single Instruction Multiple Data (SIMD) Operations

The Sharemind framework [3] for secure three-party computation showed that Single Instructions Multiple Data (SIMD) operations can result in substantial

performance gains. The idea of SIMD operations is to replace the evaluation of n identically copies of the same sub-circuit on one-bit values by one evaluation of the sub-circuit on n-bit values. This optimization reduces the overall computation time and the memory footprint as the circuit needs to be generated only once. SIMD operations are especially beneficial in data mining applications [3].

We show the benefit of SIMD operations by running the benchmarking circuit C 32 times in parallel on different inputs, resulting in a circuit C_{par} with $\mathbf{S}(C_{par}) = 32 \cdot \mathbf{S}(C) = 25{,}607{,}264$ and $\mathbf{D}(C_{par}) = \mathbf{D}(C) = 38$. The evaluation without SIMD operations required 36.44 s (i.e., amortized 1.13 s for each circuit C), of which 26.76 s (0.84 s) were spent in the setup phase and 2.87 s (0.09 s) in the online phase. Using the SIMD operations, the overall time decreased to 27.34 s (0.85 s), with similar setup time of 26.74 s (0.82 s) but 7.5 times faster online time of 0.38 s (0.012 s). In our implementation the RAM requirement of one gate without the SIMD extension is 14 Byte. C has 3,168,202 gates (including XOR gates) in total and requires 42.3 MByte of memory. To store C_{par} without the SIMD extension we therefore would need 1.32 GByte of memory. The SIMD extension increases the size per gate to 25 Byte plus one bit for each parallel exection and adds a negligible management overhead for the conversion between bitwise and SIMD operations. In total, the SIMD circuit for C_{par} requires only 88.6 MByte of memory, which corresponds to 6.5% of the non-SIMD variant.

5 Evaluation

We consider two application scenarios to evaluate the benefits of depth-optimized circuits (§3) and the optimizations of the GMW protocol (§4): privacy-preserving mobile social networks (§5.1) and privacy preserving face-recognition (§5.2).

Conceptual Performance Comparison. We first give a conceptual comparison between the GMW protocol, optimized as described in §4, with today's most efficient techniques for garbled circuits, as summarized in Tab. 3. Both techniques provide free XOR gates. Unlike garbled circuits, for each AND gate, the GMW protocol allows to shift *all* usage of symmetric cryptography into the setup phase (cf. §2.2.2) and to distribute the workload *equally* among client C and server S (cf. §4.3). Since GMW operates on single bits during the online phase, multiple gates can be evaluated in one instruction (cf. §4.4 and §4.5), whereas garbled circuits need to evaluate each gate individually. However, the total communication per AND gate for the GMW protocol is slightly larger. The memory requirement during secure evaluation of the circuit is smaller for the GMW protocol as for every wire of the circuit each party only needs to store a 1-bit share instead of an 80-bit wire label. Also the inputs are cheaper in the GMW protocol as only one random bit needs to be chosen and sent to the other party, whereas in garbled circuits a random 80-bit key needs to be chosen and sent to the evaluator (using OT for evaluator's inputs). The main disadvantage of the GMW protocol is its need for interaction during evaluation of AND gates which becomes an inevitable performance bottleneck for high network latency as shown in our practical experiments next.

Table 3. Conceptual Comparison between state-of-the-art Garbled Circuits (using Point-and-Permute, Free XORs, Garbled Row Reduction, and Pipelining) and the GMW Protocol (using optimizations of §4) for security parameter $t = 80$ bits. C: client, S: server, SHA: SHA-1, AES: AES128.

Properties	Garbled Circuits	Optimized GMW
Free XOR	yes	yes
per AND gate:		
setup computation	-	C&S: 2.5×SHA+1.8×AES
setup communication [bit]	-	C→S&C←S: 162
online computation	C: 1×SHA; S: 4×SHA	negligible
online communication [bit]	C←S: 240	C→S&C←S: 2
per wire storage C&S [bit]	80	1
per input: rnd bits | comm. [bit]	80 | C: OT resp. S: 80	1 | 1

Benchmarking Environment. We measure runtimes for different (round-trip) latencies that are typical for the following network types: LAN (0.2 ms), intra-country internet (10 ms), and trans-atlantic internet (100 ms). The latency on the network interfaces was enforced by changing the traffic control settings on Linux using the `tc` command. We used the same benchmarking environment as described in §4, but use a subgroup of order q with $|q| = 128$ and $|p| = 1024$ in the Naor-Pinkas OT to match the parameters of previous works. The time for the Naor-Pinkas OT on 1024 bit values in the LAN setting is 2,950 ms for the original implementation of [5] and 170 ms for our implementation.

5.1 Mobile Social Networks

In the mobile social network scenario of [5] one party inputs a database of N users U_i $(1 \leq i \leq N)$ that each have a set of interests H_i (represented as ℓ-bit vector) and a m-bit location L_i. The other party is a user U who can identify the user U_k that shares the most interests among all users U_i within a certain distance δ^{m+1}. As a result, U obtains in a privacy-preserving way only the identity k of U_k and 0 otherwise. We give more details on the circuit in §C.

In our experiments we use $N = 128$ users, $m = 32$-bit locations L_i, and $\ell = 255$-bit sets of interests H_i. We build two versions of the circuit, a size-efficient version MSN_S with $\mathbf{S}(\text{MSN}_S) = 89{,}321$ and $\mathbf{D}(\text{MSN}_S) = 98$ and a depth-efficient version MSN_D with $\mathbf{S}(\text{MSN}_D) = 203{,}102$ and $\mathbf{D}(\text{MSN}_D) = 68$. We compare the performance on both circuits for the original implementation of [5] to our implementation in Tab. 4.

For the LAN setting (0.2 ms) our implementation outperforms the original implementation by factor 13 (factor 3 in the online phase). Due to the multiplication triple optimization (cf. §4.1), the online time of our implementation increases only half as much as the original implementation with rising latency. While the online time of MSN_D is marginally higher than MSN_S in the LAN setting (0.2 ms), the latency becomes the dominant factor in the overall execution time for the

Table 4. Runtimes for secure evaluation of mobile social networks

Framework	[5]						This work					
Circuit	MSN_S			MSN_D			MSN_S			MSN_D		
Latency [ms]	0.2	10	100	0.2	10	100	0.2	10	100	0.2	10	100
Setup phase [s]	4.65	4.82	6.27	6.48	6.62	8.24	0.30	0.51	1.62	0.42	0.62	1.92
Online phase [s]	0.15	1.12	9.94	0.23	0.92	7.04	0.05	0.54	4.92	0.06	0.40	3.55
Overall time [s]	4.83	5.97	16.24	6.73	7.57	15.31	0.36	1.06	6.58	0.50	1.03	5.50

intra-country internet setting (10 ms) and the trans-atlantic internet setting (100 ms), i.e., MSN_D is better for settings with higher latency.

5.2 Privacy-Preserving Face Recognition

To show the performance of our framework on large circuits we benchmark it on circuits for privacy-preserving face recognition. Here, the client holds a query face and wants to determine whether it matches one of the faces input by the server. We consider two commonly used face-recognition algorithms: Eigenfaces (§5.2.1) and the index-based Hamming distance scheme of SCiFI [28] (§5.2.2).

5.2.1 Using Eigenfaces. Here, the client inputs a query face image Γ with N pixels of b-bits each. The server inputs M faces projected into a K-dimensional feature space $\Omega_1, ..., \Omega_M$. If Γ matches one of the faces in the database, the client receives the index i_{min} of the closest match (see §D for details). A privacy-preserving protocol based on additively homomorphic encryption was given in [9] and subsequently improved by combining it with garbled circuits [14, 30].

We use the same parameters as [9, 14, 30]: $N = 10{,}304$; $b = 8$; $K = 12$. Using the depth-optimized circuits of §3 results in a circuit for 320 faces in the database FR^{320}_{Eig} with $\mathbf{S}(\mathrm{FR}^{320}_{Eig}) = 14{,}887{,}713$ and $\mathbf{D}(\mathrm{FR}^{320}_{Eig}) = 120$ and a circuit for 1,000 faces FR^{1000}_{Eig} with $\mathbf{S}(\mathrm{FR}^{1000}_{Eig}) = 22{,}811{,}392$ and $\mathbf{D}(\mathrm{FR}^{1000}_{Eig}) = 128$.

The performance of our implementation in comparison with previous works is shown in Tab. 5 (similar machines connected via LAN). Our implementation outperforms previous work by at least factor 8 in the online phase for a database with 320 faces (factor 10 for 1,000 faces) while maintaining a fast setup time. Also, our implementation scales very well with increasing database size due to the SIMD operations of §4.5 (≈ 0.33 ms online time per face in the database).

5.2.2 Using Hamming Distance. In the Hamming distance based algorithm proposed in [28], the client maps his query face to an ℓ-bit index vector that represents the characteristics of the face. Face recognition is done by computing the Hamming distance between the client's index vector and each of the server's M index vectors of faces in the database and checking whether this distance is below a pre-computed, face-specific threshold. The original protocols proposed in SCiFI [28] were based on additively homomorphic encryption and later improved by the garbled circuits framework of [15].

Table 5. Runtimes for Eigenfaces-based face recognition (on similar machines)

Faces in Database	320				1,000			
Framework	[9]	[14]	[30]	This work	[9]	[14]	[30]	This work
Setup phase [s]	18	38.1	n/a	15.7	n/a	83.4	n/a	24.0
Online phase [s]	22	41.5	8.4	1.1	n/a	56.2	13	1.3
Overall time [s]	40	76.9	n/a	17.7	n/a	139.6	n/a	26.3

Table 6. Runtimes for index-based face recognition (on similar machines)

Framework	[28]	[15]				This work					
Faces in Database	100	100		320		100		320		50,000	
Latency [ms]	LAN	0.2	100	0.2	100	0.2	100	0.2	100	0.2	100
Setup phase [s]	213	0.40	0.67	0.40	0.67	0.30	1.09	0.49	1.35	44.85	55.87
Online phase [s]	31	1.75	2.18	5.14	6.34	0.006	0.53	0.01	0.62	0.90	3.02
Overall time [s]	244	8.79	9.85	42.9	44.5	0.31	1.64	0.51	2.01	45.98	59.68

We construct a circuit $\mathrm{FR}_{SCI}^{(\ell,N)}$ for the SCiFI recognition algorithm that consists of N parallel instantiations of the Boyar-Peralta count circuit CNT_{BP}^{ℓ} (cf. §3.4)[1] and the size-optimized greater than circuit $\mathrm{GT}_{S}^{\lceil \log_2 \ell+1 \rceil}$ (cf. §3.3) with $\mathbf{S}(\mathrm{FR}_{SCI}^{(\ell,N)}) = N(\ell + \lceil \log_2 \ell + 1 \rceil - d_H(\ell))$ and $\mathbf{D}(\mathrm{FR}_{SCI}^{(\ell,N)}) = \lceil \log_2 \ell + 1 \rceil + 1$. Similar to previous works we choose $\ell = 900$, resulting in a circuit with 906 AND gates per face in the database and a depth of 11. Tab. 6 shows a performance comparison between the original protocols of [28] and the framework of [15] and our framework, both evaluating the SCiFI circuit $\mathrm{FR}_{SCI}^{(900,N)}$.

The original SCiFI protocols [28] require a constant setup time of 213 s and an online time of 0.31 s per face in the database. Their main advantage is that the online phase can be parallelized on multiple servers. For the circuit-based approaches we can observe that our framework outperforms [15] in online time by factor 300 for 100 faces (factor 500 for 320 faces) in the LAN setting and by factor 4 (factor 10 for 320 faces) in the trans-atlantic internet setting. Most notable is the sub-linear scaling in the database size due to the SIMD operations (cf. §4.5), which enable our framework to process large-scale databases within a very short online time (18 μs per face in the LAN setting and 60 μs per face in the trans-atlantic internet setting). In contrast, the main performance bottleneck of the [15] framework is the time for generating large circuits, i.e., the difference between overall time and setup plus online time.

Acknowledgement. This work was supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE and by the Hessian LOEWE excellence initiative within CASED.

[1] This circuit has about half the size of the count circuit proposed in [15].

References

1. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992)
2. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995)
3. Bogdanov, D., Jagomägis, R., Laur, S.: A universal toolkit for cryptographically secure privacy-preserving data mining. In: Chau, M., Wang, G.A., Yue, W.T., Chen, H. (eds.) PAISI 2012. LNCS, vol. 7299, pp. 112–126. Springer, Heidelberg (2012)
4. Boyar, J., Peralta, R.: Concrete multiplicative complexity of symmetric functions. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 179–189. Springer, Heidelberg (2006)
5. Choi, S.G., Hwang, K.-W., Katz, J., Malkin, T., Rubenstein, D.: Secure multiparty computation of Boolean circuits with applications to privacy in on-line marketplaces. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 416–432. Springer, Heidelberg (2012)
6. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (2009)
7. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012)
8. Earle, L.G.: Latched carry-save adder. IBM Technical Disclosure Bulletin 7(10), 909–910 (1965)
9. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 235–253. Springer, Heidelberg (2009)
10. Garay, J., Schoenmakers, B., Villegas, J.: Practical and secure solutions for integer comparison. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 330–342. Springer, Heidelberg (2007)
11. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press (2004)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Symposium on Theory of Computing (STOC 1987), pp. 218–229. ACM (1987)
13. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols: Techniques and Constructions, 1st edn. Springer (2010)
14. Henecka, W., Kögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-partY computations. In: Computer and Communications Security (CCS 2010), pp. 451–462. ACM (2010)
15. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Security Symposium. USENIX (2011)
16. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
17. Kerschbaum, F.: Automatically optimizing secure computation. In: Computer and Communications Security (CCS 2011), pp. 703–714. ACM (2011)

18. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (2009)
19. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
20. Kreuter, B., Shelat, A., Shen, C.H.: Billion-gate secure computation with malicious adversaries. In: Security Symposium. USENIX (2012)
21. Ladner, R.E., Fischer, M.J.: Parallel prefix computation. Journal of the ACM 27(4), 831–838 (1980)
22. Li, B., Li, H., Xu, G., Xu, H.: Efficient reduction of 1 out of n oblivious transfers in random oracle model. Cryptology ePrint Archive, Report 2005/279 (2005)
23. Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)
24. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — a secure two-party computation system. In: Security Symposium, pp. 287–302. USENIX (2004)
25. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. Journal of Cryptology 18(1), 1–35 (2005)
26. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Electronic Commerce (EC 1999), pp. 129–139. ACM (1999)
27. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
28. Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: SCiFI - a system for secure face identification. In: Symp. on Security and Privacy, pp. 239–254. IEEE (2010)
29. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
30. Sadeghi, A.R., Schneider, T., Wehrenberg, I.: Efficient privacy-preserving face recognition. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 229–244. Springer, Heidelberg (2010)
31. Savage, J.E.: Models of Computation: Exploring the Power of Computing, 1st edn. Addison-Wesley Pub., Boston (1997)
32. Sklansky, J.: An evaluation of several two-summand binary adders. IRE Transactions on Electronic Computers EC-9(2), 213–226 (1960)
33. Yao, A.C.: How to generate and exchange secrets. In: Foundations of Computer Science (FOCS 1986), pp. 162–167. IEEE (1986)
34. Yoo, J.T., Smith, K.F., Gopalakrishnan, G.: A fast parallel squarer based on divide-and-conquer. IEEE Journal of Solid-State Circuits 32, 909–912 (1995)

A Summary of Circuit Building Blocks

Table 7. Size and Depth of Circuit Constructions (d_H: Hamming weight)

Circuit	Size **S**	Depth **D**
Addition		
Ripple-carry ADD/SUB$^{\ell}_{RC}$	ℓ	ℓ
Ladner-Fischer ADD$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + \ell$	$2\lceil\log_2 \ell\rceil + 1$
LF subtraction SUB$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + 2\ell$	$2\lceil\log_2 \ell\rceil + 2$
Carry-save ADD$^{(\ell,3)}_{CSA}$	$\ell + \mathbf{S}(\text{ADD}^{\ell})$	$\mathbf{D}(\text{ADD}^{\ell})+1$
RC network ADD$^{(\ell,n)}_{RC}$	$\ell n - \ell + n - \lceil\log_2 n\rceil - 1$	$\lceil\log_2 n - 1\rceil + \ell$
CSA network ADD$^{(\ell,n)}_{CSA}$	$\ell n - 2\ell + n - \lceil\log_2 n\rceil$ $+\mathbf{S}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$	$\lceil\log_2 n - 1\rceil$ $+\mathbf{D}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$
Multiplication		
RCN school method MUL$^{\ell}_{RC}$	$2\ell^2 - \ell$	$2\ell - 1$
CSN school method MUL$^{\ell}_{CSN}$	$2\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - \ell + 2$	$3\lceil\log_2 \ell\rceil + 4$
RC squaring SQR$^{\ell}_{RC}$	$\ell^2 - \ell$	$2\ell - 3$
LF squaring SQR$^{\ell}_{LF}$	$\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - 1.5\ell - 2$	$3\lceil\log_2 \ell\rceil + 3$
Comparison		
Equality EQ$^{\ell}$	$\ell - 1$	$\lceil\log_2 \ell\rceil$
Sequential greater than GT$^{\ell}_{S}$	ℓ	ℓ
D&C greater than GT$^{\ell}_{DC}$	$3\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil + 1$
Selection		
Multiplexer MUX$^{\ell}$	ℓ	1
Minimum MIN$^{(\ell,n)}$	$(n-1)(\mathbf{S}(\text{GT}^{\ell})+\ell)$	$\lceil\log_2 n\rceil(\mathbf{D}(\text{GT}^{\ell})+1)$
Minimum index MIN$^{(\ell,n)}_{IDX}$	$(n-1)(\mathbf{S}(\text{GT}^{\ell})+\ell+\lceil\log_2 n\rceil)$	$\lceil\log_2 n\rceil(\mathbf{D}(\text{GT}^{\ell})+1)$
Set Operations		
Set union $\cup^{\ell}$	ℓ	1
Set intersection $\cap^{\ell}$	ℓ	1
Set inclusion $\subseteq^{\ell}$	$2\ell - 1$	$\lceil\log_2 \ell\rceil + 1$
Count		
Full Adder count CNT$^{\ell}_{FA}$	$2\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil$
Boyar-Peralta count CNT$^{\ell}_{BP}$	$\ell - d_H(\ell)$	$\lfloor\log_2 \ell\rfloor$
Distances		
Manhattan distance DST$^{\ell}_{M}$	$2\mathbf{S}(\text{SUB}^{\ell})+\mathbf{S}(\text{ADD}^{(\ell,3)})+1$	$\mathbf{D}(\text{SUB}^{\ell})+\mathbf{D}(\text{ADD}^{(\ell,3)})+1$
Euclidean distance DST$^{\ell}_{E}$	$2\mathbf{S}(\text{SUB}^{\ell})+2\mathbf{S}(\text{SQR}^{\ell})$ $+\mathbf{S}(\text{ADD}^{(2\ell,4)})+2\mathbf{S}(\text{MUX}^{\ell})$	$\mathbf{D}(\text{SUB}^{\ell})$ $+\mathbf{D}(\text{SQR}^{\ell})+3$

B Depth Efficient Distance Circuits

B.1 Manhattan Distance

The Manhattan distance DST$^{\ell}_{M}$ between two points $p_1 = (x^{\ell}_1, y^{\ell}_1)$ and $p_2 = (x^{\ell}_2, y^{\ell}_2)$ is the distance in a two dimensional space allowing only horizontal and vertical moves and is computed as $|x^{\ell}_1 - x^{\ell}_2| + |y^{\ell}_1 - y^{\ell}_2|$. [5] give such a circuit DST$^{\ell}_{M,C}$ with size $\mathbf{S}(\text{DST}^{\ell}_{M,C}) = 9\ell$ and depth $\mathbf{D}(\text{DST}^{\ell}_{M,C}) = 2\ell + 2$. They use 4 multiplexer circuits MUX$^{\ell}$ (cf. [18]), 2 GT$^{\ell}_{S}$ circuits (§3.3), 2 SUB$^{\ell}_{RC}$ circuits (cf. [18]), and one ADD$^{\ell}_{RC}$ circuit (§3.1).

Optimization. We build a more efficient Manhattan distance circuit DST_M^ℓ as

$$x^{\ell+1} = \mathrm{SUB}^\ell(x_1^\ell, x_2^\ell), \qquad y^{\ell+1} = \mathrm{SUB}^\ell(y_1^\ell, y_2^\ell)$$
$$b_1^\ell = (x_{\ell+1}||x_{\ell+1}||...)^\ell \oplus (x_{\ell...1})^\ell, \quad b_2^\ell = (y_{\ell+1}||y_{\ell+1}||...)^\ell \oplus (y_{\ell...1})^\ell$$
$$\mathrm{DST}_M^\ell(p_1, p_2) = \mathrm{ADD}^{(\ell,3)}(b_1^\ell, b_2^\ell, 0^{\ell-2}||x_{\ell+1} \wedge y_{\ell+1}||x_{\ell+1} \oplus y_{\ell+1}).$$

We can choose between the size-optimized Ripple-carry and the depth-optimized Ladner-Fischer instantiations of SUB^ℓ and ADD^ℓ (§3.1). Using the Ripple-carry adder yields $\mathrm{DST}_{M,RC}^\ell$ with $\mathbf{S}(\mathrm{DST}_{M,RC}^\ell) = 4\ell + 1$ and $\mathbf{D}(\mathrm{DST}_{M,RC}^\ell) = 2\ell + 2$. The Ladner-Fischer variant $\mathrm{DST}_{M,LF}^\ell$ has approximately $\mathbf{S}(\mathrm{DST}_{M,LF}^\ell) = 3.75\ell\lceil\log_2 \ell\rceil + 7\ell + 1$ and $\mathbf{D}(\mathrm{DST}_{M,LF}^\ell) = 4\lceil\log_2 \ell\rceil + 6$.

B.2 Euclidean Distance

The Euclidean distance DST_E^ℓ between two points $p_1 = (x_1^\ell, y_1^\ell)$ and $p_2 = (x_2^\ell, y_2^\ell)$ is computed as $\sqrt{(x_1^\ell - x_2^\ell)^2 + (y_1^\ell - y_2^\ell)^2}$. Since computing the square root is very inefficient, the square of the Euclidean distance is often used instead (cf. [9]). We propose an efficient (squared) Euclidean distance circuit DST_E^ℓ as

$$x^{\ell+1} = \mathrm{SUB}^\ell(x_1^\ell, x_2^\ell), \qquad y^{\ell+1} = \mathrm{SUB}^\ell(y_1^\ell, y_2^\ell)$$
$$a^\ell = (x_{\ell+1}||x_{\ell+1}||...)^\ell \oplus (x_{\ell...1})^\ell, \qquad b^\ell = (y_{\ell+1}||y_{\ell+1}||...)^\ell \oplus (y_{\ell...1})^\ell$$
$$c^\ell = \mathrm{MUX}^\ell((0||0||...)^\ell, a^\ell, x_{\ell+1}), \qquad d^\ell = \mathrm{MUX}^\ell((0||0||...)^\ell, b^\ell, y_{\ell+1})$$
$$\mathrm{DST}_E^\ell(p_1, p_2) = \mathrm{ADD}^{(2\ell,4)}(\mathrm{SQR}^\ell(a^\ell), c^\ell||x_{\ell+1}, \mathrm{SQR}^\ell(b^\ell), d^\ell||y_{\ell+1}).$$

where $\mathrm{MUX}^\ell((0||0||...)^\ell, a^\ell, x_{\ell+1})$ is a multiplexer that selects $(0||0||...)^\ell$ if $x_{\ell+1}$ is 0, and a^ℓ else. Note that adding $c^\ell||x_{\ell+1}$ and $d^\ell||y_{\ell+1}$ in the last step can be done as part of an addition network (§3.1.2), requiring only $2\ell+2$ additional AND gates and a constant overhead in depth. DST_E^ℓ can be instantiated with depth or size efficieny in mind. The size-efficient variant $\mathrm{DST}_{E,RC}^\ell$ uses the Ripple-carry adder and has size $\mathbf{S}(\mathrm{DST}_{E,RC}^\ell) = 2\ell^2 + 6\ell + 2$ and depth $\mathbf{D}(\mathrm{DST}_{E,RC}^\ell) = 3\ell + 2$. The depth-efficient variant $\mathrm{DST}_{E,LF}^\ell$ uses the Ladner-Fischer adder and has $\mathbf{S}(\mathrm{DST}_{E,LF}^\ell) = 2\ell^2 + (7.5\ell + 2)\lceil\log_2 \ell\rceil + 11\ell - 5$ and $\mathbf{D}(\mathrm{DST}_{E,LF}^\ell) = 5\lceil\log_2 \ell\rceil + 9$.

C Mobile Social Network Circuit

The mobile social network circuit $\mathrm{MSN}^{((\ell,m),N)}$ has two steps. First, the Manhattan distance between U and each user U_i in the database is computed and compared to the threshold δ^{m+1}, resulting in a bit $c_i = \mathrm{GE}^{m+1}(\delta^{m+1}, \mathrm{DST}_M^m(L_r, L_i))$ which is used to multiplex between 0 and the size of the set intersection as s_i=$\mathrm{MUX}^{\lceil\log_2 \ell\rceil}(0^{\lceil\log_2 \ell\rceil}$,$\mathrm{CNT}^\ell(\cap^\ell(H_r, H_i)), c_i)$, where MUX is a multiplexer circuit (cf. [18]) and $\cap$ is a circuit for set-intersection (cf. [5]). Next, the index k

of the user U_k with maximum value s_i is determined using the MAX_{IDX} circuit (cf. [18]): $\text{MSN}^{((\ell,m),N)}(H_{1...N}, L_{1...N}) = \text{MAX}_{IDX}^{(\lceil \log_2 \ell \rceil, N)}(s_{1...N})$. The original circuit MSN_C in [5] has approximately size $\mathbf{S}(\text{MSN}_C^{((\ell,m),N)}) = N(10m + 3\ell + 2\lceil \log_2 \ell \rceil + \lceil \log_2 N \rceil)$ and depth $\mathbf{D}(\text{MSN}_C^{((\ell,m),N)}) = 2m + \lceil \log_2 N \rceil (\lceil \log_2 \ell \rceil + 1) + 7$.

Using the optimized circuit building blocks CNT_{BP} (§3.4), $\text{DST}_{M,RC}$ (§B.1), and GT_S (§3.3) we obtain a size-efficient variant MSN_S with approximately size $\mathbf{S}(\text{MSN}_S^{((\ell,m),N)}) = N(5m + 2\ell + 3\lceil \log_2 \ell + 1 \rceil + \lceil \log_2 N \rceil - d_H(\ell) + 2)$ and depth $\mathbf{D}(\text{MSN}_S^{((\ell,m),N)}) = 2m + \lceil \log_2 N \rceil (\lceil \log_2 \ell + 1 \rceil + 1) + 4$. Alternatively, using $\text{DST}_{M,LF}$ (§B.1) and GE_{DC} (§3.3) yields a depth-efficient variant MSN_D which has size $\mathbf{S}(\text{MSN}_D^{((\ell,m),N)}) = N(3.75m\lceil \log_2 m \rceil + 9m + 3\ell + 4\lceil \log_2 \ell + 1 \rceil + \lceil \log_2 N \rceil)$ and depth $\mathbf{D}(\text{MSN}_D^{((\ell,m),N)}) = 5\lceil \log_2 m \rceil + \lceil \log_2 N \rceil (\lceil \log_2 \lceil \log_2 \ell + 1 \rceil \rceil + 2) + 8$. The difference between the two circuits is that the size of the depth-efficient circuit is larger by approximately $N(3.75m\lceil \log_2 m \rceil + 4m + \ell)$ whereas its depth decreases logarithmically in m and $\log_2 \ell$.

D Face Recognition Circuit

The circuit for face recognition FR implements the Eigenface recognition algorithm (cf. [30] for a detailed description) by first projecting the face image Γ into the K-dimensional Eigenface space. The projection is performed by computing for $i = 1, \ldots, K$: $\omega_i = \sum_{j=1}^{N} u_{i,j}\Gamma_j - \sum_{j=1}^{N} u_{i,j}\Psi_j$ where the server inputs the Eigenfaces $u_{i,j}$ and the average face Ψ and locally precomputes $-\sum_{j=1}^{N} u_{i,j}\Psi_j$. Afterwards, the square of the Euclidean distance between the ω_i and the server's projected faces $\Omega_1, ..., \Omega_M$ is computed as $D_j = \sum_{i=1}^{K} (\Omega_{j,i} - \omega_i)^2$, for $j = 1, \ldots, M$. Finally, the minimum distance D_{min} is selected and compared to a pre-determined threshold τ, input by the server. If $D_{min} \leq \tau$ the client learns j_{min} and $\perp$ otherwise.

As squaring is cheaper than multiplying (cf. §3.2), we optimize computation of the projection phase by computing $\sum_{j=1}^{N} u_{i,j}\Gamma_j$ as $\sum_{j=1}^{N} \frac{(u_{i,j}+\Gamma_j)^2 - u_{i,j}^2 - \Gamma_j^2}{2}$. Although this form might look more complex at a first glance, the resulting circuit requires only around half the number of AND gates compared to [30]. Using the optimization of [17], the server locally computes and inputs $-\sum_{j=1}^{N} u_{i,j}^2$ and the client locally computes and inputs $-\sum_{j=1}^{N} \Gamma_j^2$.

As building blocks we use a generalization of the Euclidean distance circuit DST_E (§B.2) to K dimensions, ADD_{CSA} and ADD_{RC} (§3.1), GT_{DC} (§3.3), and a circuit for computing the minimum index $\text{MIN}_{IDX,DC}$ (cf. [18]).