Efficiently Outsourcing Multiparty Computation under Multiple Keys

Andreas Peter, Erik Tews, and Stefan Katzenbeisser

Security Engineering Group
Technische Universität Darmstadt and CASED, Germany
{peter,e_tews,katzenbeisser}@seceng.informatik.tu-darmstadt.de

Abstract. Secure Multiparty Computation (SMC) enables a set of users to evaluate certain functionalities on their respective inputs while keeping these inputs encrypted throughout the computation. In many scenarios, however, outsourcing these computations to an untrusted server is desirable, so that the server can perform the computation on behalf of the users. Unfortunately, existing solutions are either inefficient, rely heavily on user interaction, or require the inputs to be encrypted under the *same* key—drawbacks making the employment in practice very limited. We propose the first general-purpose construction that avoids all these drawbacks: it is efficient, it requires no user interaction whatsoever (except for data up- and download), and it allows evaluating any dynamically chosen function on inputs encrypted under *different* independent public keys. Our solution assumes the existence of two non-colluding but untrusted servers that jointly perform the computation by means of a cryptographic protocol. This protocol is provably secure in the semi-honest model. We demonstrate the applicability of our result in two real-world scenarios from different domains: Privacy-Preserving Face Recognition and Private Smart Metering. Finally, we give a performance analysis of our general-purpose construction to highlight its practicability.

Keywords: Secure Multiparty Computation, Homomorphic Encryption, Outsourcing Computation, Semi-Honest Model

1 Introduction

Online communication in today's society is mostly being done through central web-servers which process vast amounts of private data. Examples of online services exploiting this paradigm are social networks, online auctions, and cloud services to name just a few. In the recent years, many concerns have been raised regarding data privacy in these scenarios, and serious privacy breaches have occurred [2, 4, 6, 51].

In order to deal with these privacy threats to sensitive data, the concept of Secure Multiparty Computation (SMC) gains increasing importance. In this setting, the computation is carried out interactively between several participating parties, in a way that sensitive data is kept hidden (for example encrypted or shared among protocol participants) and only the desired output of the computation is available. In this paper we focus on solutions based on homomorphic encryption [19, 37]. All current practically feasible SMC solutions heavily rely on interaction, since the basic encryption schemes in use support only limited homomorphisms. In order to perform more complex operations, decryption steps are needed, which require parties holding a secret key, or a share thereof, to be online. This interactive nature of the protocols greatly hinders adoption. Consider, for example, an application scenario where a number of clients encrypt individual input data (such as individual sales records) and push it to a server, which is supposed to aggregate the data (in order to compute global sales statistics). In this case it is not feasible to assume that all (or most) clients are still online and able to assist the server in its computations.

The same problem holds for other scenarios as well, such as privacy-preserving smart metering: while individual meters should be able to encrypt their data and push it to a server for further processing, the party who performed the initial encryption should not be involved in further computations. This application shows another key characteristic of most practical problems: rather than encrypting all input data with the *same* public key, which is required by all known efficient SMC solutions, each party should be able to use *its own* pair of public and private keys. Thus, an efficient SMC solution is required that *limits interaction* with clients as much as possible, while allowing to compute on data encrypted with *different public keys*. More precisely, we are considering the following scenario in this paper:

1. A set of n mutually distrusting clients $P_1, \dots, P_n$ (the number n may change dynamically over time), each having its own public and private key pair, encrypt data under their respective public keys and store these encryptions on a server $\mathcal{C}$.
2. Any *dynamically chosen* function (i.e., the function does not need to be specified at the time data is encrypted) should be computed by $\mathcal{C}$ on the clients' data, while all inputs and intermediate results remain private.
3. Due to the fact that clients are not always online in practice, $\mathcal{C}$ needs the ability to compute these functions *without any interaction* of the clients. In particular, this also concerns the clients' retrieval of results.
4. Once online, individual clients can retrieve the result while the server learns nothing at all.

We present a simple and efficient solution to this scenario, meaning that we give a general-purpose cryptographic protocol (with no client interaction) that allows outsourcing of any multi-party computation of inputs encrypted under multiple unrelated public keys. Our solution employs two non-colluding, semi-honest but untrusted servers $\mathcal{C}$ and $\mathcal{S}$. All steps in our protocol rely on *no interaction* with the clients whatsoever. These clients only initially store their encrypted inputs on the (main) server $\mathcal{C}$ who in turn can compute *any dynamically chosen* n-input function on n given (encrypted) inputs in an SMC protocol together with the second server $\mathcal{S}$. We show our protocols secure in the semi-honest model, meaning that all protocol participants follow the protocol description, but may try to gather information about other parties' inputs, intermediate results, or overall outputs just by looking at the transcripts. For performance reasons, this is the predominant security model used in practical implementations of SMC [11, 16, 24, 28, 43, 50, 23], which particularly makes sense in our setting where the servers (performing the computations) are business driven parties in practice, and cheating would cause negative publicity and harm their reputation.

Assuming the existence of two non-colluding servers in order to perform the secure computations is very common both in the theoretical (e.g., [21, 18]) and the practical community (e.g., [16, 24]). In fact, according to [55], a completely non-interactive solution in the *single* server setting can be proven to be impossible to realize (drawing on the impossibility of program obfuscation). Hence, if complete non-interaction of the clients is required (as in our case), we need at least two servers and so our solution is the best we can hope for in this regard. Moreover, several real-world applications relied on the assumption of multiple non-colluding servers in the past [11, 16, 10]. For instance when considering cloud computing scenarios, the multiple servers could be different cloud providers [46].

We make extensive use of the BCP cryptosystem by Bresson, Catalano and Pointcheval [15] which is both additively homomorphic (i.e., it allows addition of plaintexts in the encrypted domain) and offers two independent decryption mechanisms. The successful usage of the second decryption mechanism depends on a master secret key that is stored on the second server $\mathcal{S}$ in our proposal. With this in mind, the basic idea of our construction consists of three steps:

1. After collecting the individually encrypted inputs, the main server $\mathcal{C}$ runs an SMC protocol with $\mathcal{S}$ in order to transform the given inputs (encrypted under the clients' public keys) into encryptions under the *product* of all involved public keys (ensuring that *all* clients need to participate in a successful decryption) without changing the underlying plaintexts.
2. With these transformed ciphertexts (under the *same* public key), we can run traditional addition and multiplication SMC protocols by using the additively homomorphic property of the underlying cryptosystem, allowing to compute any function represented by an arithmetic circuit.
3. Once the result (encrypted under the product of all keys) is ready, $\mathcal{C}$ runs a final SMC protocol with $\mathcal{S}$ in order to transform this result back into encryptions under the clients' respective public keys.

Finally, we show the applicability of our framework to two application scenarios taken from different domains. Recently, SMC protocols have been used to construct privacy-preserving protocols for face recognition [23, 52]. We show how our general-purpose protocols can be leveraged here in order to get rid of interaction with the users of such systems. To highlight the real-world applicability of our construction, we elaborate on several scenarios in the area of social networks where face recognition is used for image tagging services [5] or in order to help law enforcement agencies to prosecute suspected persons [3]. Since social network users usually access their profiles via resource-constrained mobile devices that are not always online, a completely non-interactive solution is crucial for such systems to work in practice. We adapt the privacy-preserving face recognition protocol presented in [23] to our construction and show its practicability by giving an implementation together with a detailed performance analysis. Similarly, privately collecting sensor readings from smart meters has been an important application of SMC protocols recently [28, 42, 43]. We show that certain variants of our original proposal can be used to efficiently enhance privacy in smart grids. For instance, we design a protocol to privately aggregate sensor readings with very low computational costs at the smart meters due to our non-interactive construction. Additionally, we can realize privacy-preserving billing under complex policies.

Related Work. Previous papers on SMC protocols were concerned with *interactive* solutions where all parties are actively involved in computing an arbitrary function on their respective inputs in a privacy-preserving manner [17, 32, 44, 50, 56, 36, 22, 9]. Since we strive for a non-interactive solution, these constructions are not applicable in our scenario. To reduce computational costs at the clients' side, SMC has been considered in the client/server model (as we do) [11, 21, 25, 35, 39, 48, 40], but again with interaction of the clients during and/or after the computations. Furthermore, Choi et al. [18] give a solution with minimal interaction of the clients, while relying on two non-colluding servers (as in our construction). Their solution, however, is mostly of theoretical interest both for efficiency reasons and because clients are bound to encrypt their private inputs under a *single* public key that is shared between the two servers (so clients do not have individual private keys). Therefore, in order to efficiently deal with modern star-like communication patterns where clients store their data on a central server (encrypted under their own associated public keys), Halevi et al. [33] proposed a solution in which, although being non-interactive, the server is entitled to learn the result of the computation which contradicts our setting where only clients are allowed to learn the output.

In [29], Gentry proposes to leverage Fully Homomorphic Encryption (FHE) to solve the problem statement in our setting. Unfortunately, besides the lack of efficiency of recent FHE schemes [12–14, 30], the main drawback is that all clients need to run an interactive setup and an interactive decryption phase after the server computed the result.

Very recently, López-Alt et al. [45] introduced the notion of *On-the-Fly* SMC as the first solution to a similar scenario as ours, yet still relying on an interactive decryption phase. They implement this by using a novel primitive called *Multikey* FHE that allows computation on data encrypted under multiple unrelated public keys, having similar efficiency shortcomings as all the other FHE schemes.

In a nutshell, all existing solutions are not applicable in our setting where SMC is to be performed on data encrypted under multiple unrelated keys, except for one [45] which lacks in efficiency and relies on an interactive user decryption.

Outline. Some standard notation, the security model of semi-honest adversaries, and the BCP encryption scheme are summarized in Section 2. We give our construction and proofs of correctness for the individual building blocks in Section 3, while analyzing security in Section 4. Useful variants and opimizations of our protocols are given in Section 5. We summarize our experimental results in Section 6 and elaborate on application scenarios in Section 7. Finally, we conclude in Section 8 while focusing on possible future work.

2 Preliminaries

2.1 Notation and Security Model

Throughout the paper, we use the following standard notation: We write $x \longleftarrow X$ if X is a random variable or distribution and x is to be chosen randomly from X according to its distribution. In the case where X is solely a set, $x \overset{U}{\longleftarrow} X$ denotes that x is chosen uniformly at random from X. For an algorithm $\mathcal{A}$ we write $x \longleftarrow \mathcal{A}(y)$ if $\mathcal{A}$ outputs x on fixed input y according to $\mathcal{A}$'s distribution.

We assume all participants to be honest-but-curious, i.e., we consider the semi-honest model [31]. This means that all involved parties follow the protocols, but try to gather information about the outputs (or intermediate results) of the computation just by looking at the protocols' transcripts. In addition, we assume that there is no collusion between any of the parties. Considering this model makes sense in our scenario as servers performing the computations are business driven parties in practice who do not want to harm their reputation and therefore avoid cheating (which would cause negative publicity).

From a theoretical point of view, designing protocols in the semi-honest model is considered as the first step towards protocols that can deal with malicious adversaries. We see this as future work.

2.2 Additively Homomorphic Encryption

A public-key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *additively homomorphic* if there is an operation "$\cdot$" in the encrypted domain (usually this is the multiplication) such that for given ciphertexts c_1 and c_2, it holds that $c_1 \cdot c_2$ is an encryption of the addition of the underlying plaintexts:

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2)) = m_1 + m_2, \tag{1}$$

where pk and sk are the public and secret key, respectively, and m_1, m_2 are two plaintext messages. For a more extensive and formal treatment, we refer to Armknecht et al. [7].

In this work, we use the additively homomorphic cryptosystem by Bresson, Catalano and Pointcheval (BCP) [15] which additionally offers two independent decryption mechanisms. The second decryption mechanism decrypts a given ciphertext successfully if and only if a certain master secret key is known. The general setting for such schemes with a double decryption mechanism is as follows: Besides the usual key generation, encryption and decryption algorithms of the users, a *master* runs an initial setup to generate *public parameters* and a *master secret*. This master secret can then be used by the master in a *master decryption* algorithm (the second decryption mechanism) to successfully decrypt *any* given ciphertext.

We require such homomorphic schemes with a double decryption mechanism to be *semantically secure*, which informally means that one cannot distinguish between encryptions of known messages and random messages. More details and stronger security notions can be found in [26].

The BCP Cryptosystem and its Properties. Essentially, the BCP cryptosystem [15] is an additively homomorphic variant of the El Gamal cryptosystem [27]:

$\mathsf{Setup}(\kappa)$: For a security parameter κ, choose a safe-prime RSA-modulus $N = pq$ (i.e., $p = 2p' + 1$ and $q = 2q' + 1$ for distinct primes p' and q', respectively) of bitlength κ. Pick a random element $g \in \mathbb{Z}^*_{N^2}$ of order $pp'qq'$ such that $g^{p'q'} \mod N^2 = 1 + kN$ for $k \in [1, N-1]$. The plaintext space is $\mathbb{Z}_N$ and the algorithm outputs

$$\begin{array}{ll} \textit{public parameters} & \mathsf{PP} = (N, k, g) \\ \textit{master secret} & \mathsf{MK} = (p', q'). \end{array} \tag{2}$$

$\mathsf{KeyGen}(\mathsf{PP})$: Pick a random $a \in \mathbb{Z}_{N^2}$ and compute $h = g^a \mod N^2$. The algorithm outputs the

$$\textit{public key}\ \mathsf{pk} = h \text{ and } \textit{secret key}\ \mathsf{sk} = a. \tag{3}$$

$\mathsf{Enc}_{(\mathsf{PP},\mathsf{pk})}(m)$: Given a plaintext $m \in \mathbb{Z}_N$, pick a random $r \in \mathbb{Z}_{N^2}$ and output the ciphertext (A, B) as

$$A = g^r \mod N^2 \quad B = h^r(1 + mN) \mod N^2. \tag{4}$$

$\mathsf{Dec}_{(\mathsf{PP},\mathsf{sk})}(A,B)$: Given a ciphertext (A,B) and secret key $\mathsf{sk} = a$, output the plaintext m as

$$m = \frac{B/(A^a) - 1 \mod N^2}{N}. \tag{5}$$

$\mathsf{mDec}_{(\mathsf{PP},\mathsf{pk},\mathsf{MK})}(A,B)$: Given a ciphertext (A,B) (encrypted using the randomness $r \in \mathbb{Z}_{N^2}$), a user's public key $\mathsf{pk} = h$ and the master secret MK. Let $\mathsf{sk} = a$ denote the user's private key corresponding to $\mathsf{pk} = h$. First compute $a \mod N$ as

$$a \mod N = \frac{h^{p'q'} - 1 \mod N^2}{N} \cdot k^{-1} \mod N, \tag{6}$$

where k^{-1} denotes the inverse of k modulo N. Then compute $r \mod N$ as

$$r \mod N = \frac{A^{p'q'} - 1 \mod N^2}{N} \cdot k^{-1} \mod N. \tag{7}$$

Let δ denote the inverse of $p'q'$ modulo N and set $\gamma := ar \mod N$. The algorithm outputs the plaintext m as

$$m = \frac{(B/(g^{\gamma}))^{p'q'} - 1 \mod N^2}{N} \cdot \delta \mod N. \tag{8}$$

For proofs of correctness and semantic security (under the Decisional Diffie-Hellman assumption), we refer to [15]. If the context is clear, we omit the public parameters PP in the algorithms, e.g., we write $\mathsf{Enc}_{\mathsf{pk}}(m)$ instead of $\mathsf{Enc}_{(\mathsf{PP},\mathsf{pk})}(m)$.

3 Our Construction

Recall that we are considering a scenario with n (mutually distrusting) clients, denoted by $P_1, \ldots, P_n$, each having its own pair of public and private keys $(\mathsf{pk}_i, \mathsf{sk}_i)$, $i = 1, \ldots, n$.[1] Each client P_i $(i = 1, \ldots, n)$ stores private data m_i encrypted under its respective public key pk_i on an untrusted server $\mathcal{C}$.

Now, $\mathcal{C}$ is assigned to compute an arbitrary n-input function f on the clients' inputs, while keeping the inputs and intermediate results private. We represent such functions f by means of arithmetic circuits, i.e., the computation of a given function amounts to the evaluation of addition and multiplication gates over encrypted inputs. More details on this and other means of representing a function f can be found in [41].

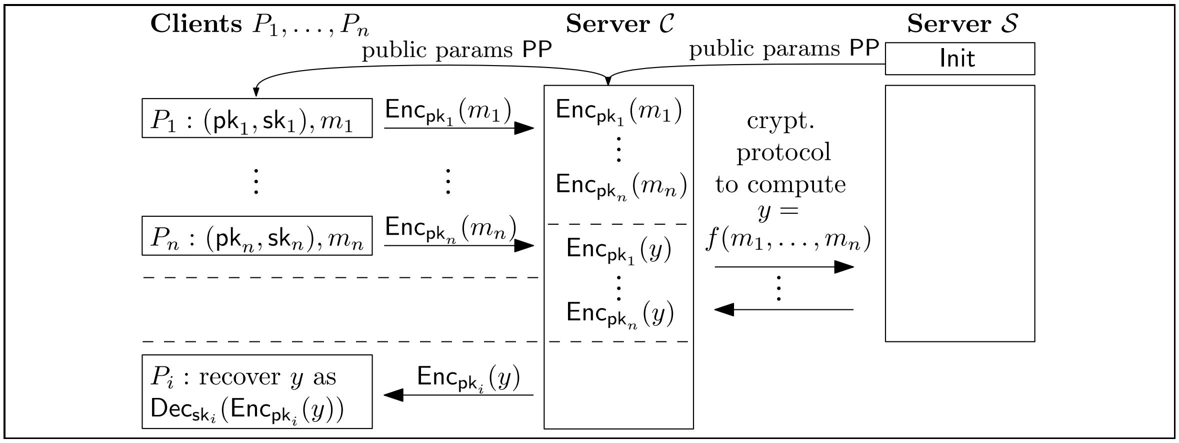
Our basic idea to realize this functionality can be summarized as follows (illustrated in Figure 1):

1. We assume the existence of a second untrusted server $\mathcal{S}$ that acts semi-honestly and that does not collude with any of the other parties. See the discussion on the semi-honest model in Section 2 and on the use of two non-colluding servers in the Introduction for further details on why this is a reasonable and worthwhile assumption.

[1] Fixing the number n initially is for reasons of readability only. In fact, in our construction, this number is allowed to change over time. More importantly, clients are able to generate their own pair of public and private keys without communicating to some trusted third party. Therefore, participation in the system is a dynamic process.

2. Initially, this second server $\mathcal{S}$ runs a setup Init that sets up the system and distributes the system's public parameters.
3. After this initial setup, clients can use the cryptosystem's KeyGen (independently of any further party) to generate their respective pair of public and private keys, and to upload encryptions of their private data to the first server $\mathcal{C}$.
4. Once an (arbitrary) function f is to be evaluated on the, say, n inputs $m_1, \ldots, m_n$ of clients $P_1, \ldots, P_n$, the server $\mathcal{C}$ runs a cryptographic protocol with the second server $\mathcal{S}$ that consists of only four building blocks: $\mathsf{KeyProd}, \mathsf{Add}, \mathsf{Mult}$ and $\mathsf{TransDec}$. $\mathsf{KeyProd}$ tranforms all ciphertexts to encryptions under a single public key (whose corresponding secret key is unknown), Add and Mult evaluate addition and multiplication gates on encrypted inputs, respectively, and $\mathsf{TransDec}$ transforms the encrypted result $f(m_1, \ldots, m_n)$ back to n encryptions each under a different client's public key.

   The overall protocol is run with no interaction of the clients whatsoever. Recall that the inputs $m_1, \ldots, m_n$ are encrypted under *different* public keys.
5. After all computations are done, each client retrieves the encrypted output of the server $\mathcal{C}$ which it decrypts locally with its respective private key in order to get the result $f(m_1, \ldots, m_n)$.



Clients $P_1, \ldots, P_n$
Server $\mathcal{C}$
Server $\mathcal{S}$
public params PP
public params PP
Init
$P_1 : (\mathsf{pk}_1, \mathsf{sk}_1), m_1$
$\mathsf{Enc}_{\mathsf{pk}_1}(m_1)$
$\mathsf{Enc}_{\mathsf{pk}_1}(m_1)$
$\mathsf{Enc}_{\mathsf{pk}_n}(m_n)$
crypt. protocol to compute $y = f(m_1, \ldots, m_n)$
$P_n : (\mathsf{pk}_n, \mathsf{sk}_n), m_n$
$\mathsf{Enc}_{\mathsf{pk}_n}(m_n)$
$\mathsf{Enc}_{\mathsf{pk}_1}(y)$
$\mathsf{Enc}_{\mathsf{pk}_n}(y)$
P_i : recover y as $\mathsf{Dec}_{\mathsf{sk}_i}(\mathsf{Enc}_{\mathsf{pk}_i}(y))$
$\mathsf{Enc}_{\mathsf{pk}_i}(y)$

Fig. 1. The basic concept of our construction.

In the following, we explain the individual steps of our protocols:

Initialization. Initially, a setup process initializes the BCP cryptosystem and distributes the system's public parameters. This setup is run by the second server $\mathcal{S}$ (since $\mathcal{S}$ is semi-honest and needs the master secret). We denote this algorithm by Init, which simply runs the algorithm Setup of the BCP cryptosystem and sends its public parameters $\mathsf{PP} = (N, k, g)$ to the server $\mathcal{C}$ (see Figure 2).

Data Upload. In order to upload private data to the server $\mathcal{C}$, a client P first needs to receive the system's public parameters $\mathsf{PP} = (N, k, g)$ to be able to generate its own pair of public and private keys. After these keys are generated, the client P can encrypt its private data using Enc and upload it together with its public key to the server $\mathcal{C}$. The details of this procedure can be found in Figure 3.

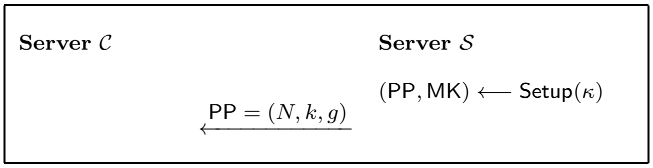

Server $\mathcal{C}$
Server $\mathcal{S}$
$(\mathsf{PP}, \mathsf{MK}) \longleftarrow \mathsf{Setup}(\kappa)$
$\xleftarrow{\mathsf{PP} = (N, k, g)}$

Fig. 2. $\mathsf{Init}(\kappa)$. The initial setup algorithm, where κ is the security parameter for the BCP scheme.

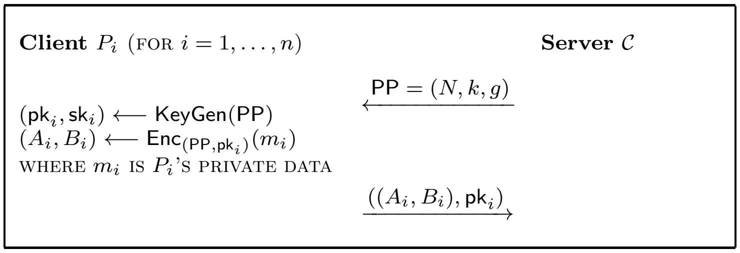

Client P_i (FOR $i = 1, \ldots, n$)
Server $\mathcal{C}$
$\xleftarrow{\mathsf{PP} = (N, k, g)}$
$(\mathsf{pk}_i, \mathsf{sk}_i) \longleftarrow \mathsf{KeyGen}(\mathsf{PP})$
$(A_i, B_i) \longleftarrow \mathsf{Enc}_{(\mathsf{PP}, \mathsf{pk}_i)}(m_i)$
WHERE m_i IS P_i'S PRIVATE DATA
$\xrightarrow{((A_i, B_i), \mathsf{pk}_i)}$

Fig. 3. Steps that have to be taken by a client P_i (for $i = 1, \ldots, n$) which wants to participate in computations made on the server $\mathcal{C}$.

Cryptographic Protocol between Servers $\mathcal{C}$ and $\mathcal{S}$. Assume that the server $\mathcal{C}$ wants to compute an encryption of $f(m_1, \ldots, m_n)$ for an n-input function f where $m_1, \ldots, m_n$ are the private inputs of the clients $P_1, \ldots, P_n$. Recall that during the data upload phase, $\mathcal{C}$ retrieved only encryptions of the inputs $m_1, \ldots, m_n$. $\mathcal{C}$ does its computations by means of a cryptographic protocol between $\mathcal{C}$ and $\mathcal{S}$ consisting of the 4 subprotocols: $\mathsf{KeyProd}$, Add, Mult and $\mathsf{TransDec}$.

Recall that we represent the function f by an arithmetic circuit, meaning that we have to be able to securely evaluate addition and multiplication gates. Addition gates seem to be easy to deal with since the underlying cryptosystem is additively homomorphic. Unfortunately though, the clients' inputs are encrypted under *different* public keys and the additive property of the BCP cryptosystem only works for encryptions under the *same* public key. Therefore, the server $\mathcal{C}$ first runs the algorithm $\mathsf{KeyProd}$ which transforms all involved ciphertexts to encryptions under a *single* key. This single key is the product of all involved public keys and so $\mathcal{C}$ remains unable to decrypt the ciphertexts as it does not know the corresponding secret key. In fact, the secret key needed to decrypt encryptions under the product of all clients' public keys is the sum of all clients' secret keys. Of course, decryption still works by using the master secret which is only known to the second server $\mathcal{S}$ and *not* to $\mathcal{C}$. We stress that $\mathcal{S}$ never gets to see encryptions of the clients' original inputs but only blinded versions of them, so it does not learn these inputs although having the master secret.

After this key-transformation of ciphertexts, the additive property of the underlying cryptosystem can be exploited to securely evaluate addition gates. This step is denoted by Add in our construction. Multiplication gates can be securely evaluated by (an adapted version of) the well-known protocol of [19], essentially relying on "blinding-the-plaintext" techniques. This is done by our protocol Mult.

Finally, once the complete arithmetic circuit representing the function f is successfully evaluated by using Add and Mult, $\mathcal{C}$ runs the protocol $\mathsf{TransDec}$ in order to transform the results (encrypted under the product of all public keys) back to encryptions of the individual clients' public keys without changing the underlying plaintext.

In the following, we describe the individual building blocks for this protocol between $\mathcal{C}$ and $\mathcal{S}$.

The Subprotocol KeyProd. The purpose of this protocol is to transform the encryptions of all participating clients $P_1, \ldots, P_n$ into encryptions under a single public key, namely the product $\mathsf{Prod.pk} := \prod_{i=1}^{n} \mathsf{pk}_i \mod N^2$ of all involved public keys without changing the underlying plaintexts. Using the product key here ensures that it is not a key of $\mathcal{C}$'s choosing. In fact, the corresponding secret key required to successfully decrypt an encryption under $\mathsf{Prod.pk}$ is the sum $\sum_{i=1}^{n} \mathsf{sk}_i$ of all clients' secret keys. This subprotocol needs to be run only *once* per fixed set of encrypted inputs and does not depend on the actual function $\mathcal{C}$ wants to evaluate. This means that after the execution of KeyProd, any function can be computed on the transformed ciphertexts.

For a given ciphertext (A_i, B_i) encrypted under the public key pk_i of the client P_i $(i = 1, \ldots, n)$, $\mathcal{C}$ blinds the ciphertext with a random message τ_i and sends it to $\mathcal{S}$. Since $\mathcal{S}$ knows the master secret MK, it uses it to decrypt this blinded ciphertext and re-encrypt it under the product of all clients' public keys. The result of this is then sent back to $\mathcal{C}$ who can remove the blinding τ_i again, achieving an encryption under the product key without changing the underlying plaintext. A detailed description of these steps can be seen in Figure 4.

```
Server $\mathcal{C}$, STORES $((A_i, B_i), \mathsf{pk}_i), i = 1, \ldots, n$          Server $\mathcal{S}$, STORES $\mathsf{MK} = (p', q')$

ON INPUT CIPHERTEXTS $(A_1, B_1), \ldots, (A_n, B_n)$:
$\mathsf{Prod.pk} \longleftarrow \prod_{i=1}^{n} \mathsf{pk}_i \mod N^2$
FOR ALL $i = 1, \ldots, n$:
  PICK RANDOM $\tau_i \overset{U}{\longleftarrow} \mathbb{Z}_N$
  $(C_i, D_i) \longleftarrow \mathsf{Add}((A_i, B_i), \mathsf{Enc}_{\mathsf{pk}_i}(\tau_i))$
                              $\xrightarrow[(\mathsf{pk}_i, (C_i, D_i))]{\forall i = 1, \ldots, n :}$
                                                        $\mathsf{Prod.pk} \longleftarrow \prod_{i=1}^{n} \mathsf{pk}_i \mod N^2$
                                                        FOR ALL $i = 1, \ldots, n$:
                                                          $z_i \longleftarrow \mathsf{mDec}_{(\mathsf{pk}_i, \mathsf{MK})}(C_i, D_i)$
                                                          $(W_i, Z_i) \longleftarrow \mathsf{Enc}_{\mathsf{Prod.pk}}(z_i)$
                              $\xleftarrow[(W_i, Z_i)]{\forall i = 1, \ldots, n :}$
FOR ALL $i = 1, \ldots, n$:
  $(\overline{A_i}, \overline{B_i}) \longleftarrow \mathsf{Add}((W_i, Z_i), \mathsf{Enc}_{\mathsf{Prod.pk}}(-\tau))$
OUTPUT THE CIPHERTEXTS $(\overline{A_1}, \overline{B_1}), \ldots, (\overline{A_n}, \overline{B_n})$
```

Fig. 4. KeyProd. Transforms encryptions under $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ into encryptions under $\mathsf{Prod.pk} = \prod_{i=1}^{n} \mathsf{pk}_i \mod N^2$ without changing the underlying plaintexts.

The Subprotocols Add and Mult. Recall that the server $\mathcal{C}$ wants to compute the function f, which we consider to be represented as an arithmetic circuit over the ring $\mathbb{Z}_N$ (note that hitting on a value in $\mathbb{Z}_N$ which is not invertible modulo N happens with negligible probability only). Therefore, we have to deal with addition- and multiplication-gates in $\mathbb{Z}_N$. Without loss of generality, we consider these as 2-input-1-output gates. The algorithm Add deals with an addition-gate, while the subprotocol Mult deals with a multiplication-gate. We start with the former and stress that it is a non-interactive protocol which does not need the server $\mathcal{S}$. This is due to the fact that the underlying BCP cryptosystem is additively homomorphic for encryptions under the *same* public key. We recall that this is exactly what the subprotocol KeyProd achieved by computing encryptions under the product Prod.pk, so all clients' private inputs $m_1, \ldots, m_n$ are now encrypted under the same public key. The algorithm to perform the addition is depicted in Figure 5.

```
Server $\mathcal{C}$ stores encryptions
  of the form $(A, B)$ encrypted under $\mathsf{Prod.pk}$

On input two ciphertexts $(A, B)$ and $(A', B')$:
  $(\overline{A}, \overline{B}) \longleftarrow (A \cdot A' \mod N^2, B \cdot B' \mod N^2)$
output the ciphertext $(\overline{A}, \overline{B})$
```

Fig. 5. Add. Given two ciphertexts (A, B) and (A', B') encrypted under Prod.pk, it computes an encryption of the sum of the underlying plaintexts.

A multiplication-gate, however, has to be computed interactively with the server $\mathcal{S}$. In fact, the protocol we use is an adaptation of the well-known multiplication protocol of [19] which sends blinded version of the original ciphertexts (that are to be multiplied) to $\mathcal{S}$ that in turn uses the master secret to decrypt. Then, $\mathcal{S}$ performs the multiplication in the clear and re-encrypts. The (encrypted) result is sent back to $\mathcal{C}$, which can remove the blinding again. Details are given in Figure 6.

Server $\mathcal{C}$, stores $\mathsf{Prod.pk}$ and $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$		**Server** $\mathcal{S}$, stores $\mathsf{MK}, \mathsf{Prod.pk}$, and $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$
On input ciphertexts (A, B) and (A', B') under $\mathsf{Prod.pk}$:		
Pick random $\sigma, \sigma' \overset{U}{\longleftarrow} \mathbb{Z}_N$		
$(C, D) \longleftarrow \mathsf{Add}((A, B), \mathsf{Enc}_{\mathsf{Prod.pk}}(-\sigma))$		
$(C', D') \longleftarrow \mathsf{Add}((A', B'), \mathsf{Enc}_{\mathsf{Prod.pk}}(-\sigma'))$	$\xrightarrow{(C, D), (C', D')}$	
		$z \longleftarrow \mathsf{mDec}_{(\mathsf{Prod.pk}, \mathsf{MK})}(C, D)$
		$z' \longleftarrow \mathsf{mDec}_{(\mathsf{Prod.pk}, \mathsf{MK})}(C', D')$
	$\xleftarrow{(Z_1, Z_2)}$	$(Z_1, Z_2) \longleftarrow \mathsf{Enc}_{\mathsf{Prod.pk}}(z \cdot z')$
$(T_1, T_2) \longleftarrow \mathsf{Enc}_{\mathsf{Prod.pk}}(-\sigma\sigma')$		
$(\overline{A}, \overline{B}) \longleftarrow (Z_1 A^{\sigma'} A'^{\sigma} T_1 \mod N^2, Z_2 B^{\sigma'} B'^{\sigma} T_2 \mod N^2)$		
Output ciphertext $(\overline{A}, \overline{B})$		

Fig. 6. Mult. Given two ciphertexts (A, B) and (A', B') encrypted under Prod.pk, it computes an encryption of the multiplication of the underlying plaintexts.

The Subprotocol TransDec. Finally, the task of subprotocol TransDec is to take the encrypted result of $f(m_1, \ldots, m_n)$, encrypted under Prod.pk, and to transform it back to n encryptions of the same plaintext $f(m_1, \ldots, m_n)$, each under a different client's public key $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, respectively.

Again, the idea is to blind the original ciphertext and send it to $\mathcal{S}$, which in turn decrypts using the master secret and then creates n encryptions for each client's public key. The created ciphertexts are returned to $\mathcal{C}$ which removes the blindings. The precise steps of this protocol are summarized in Figure 7.

Server $\mathcal{C}$, stores $\mathsf{Prod.pk}$ and $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$		**Server $\mathcal{S}$**, stores $\mathsf{MK}, \mathsf{Prod.pk}$ and $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$
On input a ciphertext (A, B) under $\mathsf{Prod.pk}$: Pick random $\tau \overset{U}{\longleftarrow} \mathbb{Z}_N$ $(C, D) \longleftarrow \mathsf{Add}((A, B), \mathsf{Enc}_{\mathsf{Prod.pk}}(\tau))$	$\xrightarrow{(C, D)}$	
		$z \longleftarrow \mathsf{mDec}_{(\mathsf{Prod.pk},\mathsf{MK})}(C, D)$ for all $i = 1, \ldots, n$: $(W_i, Z_i) \longleftarrow \mathsf{Enc}_{\mathsf{pk}_i}(z)$
for all $i = 1, \ldots, n$: $(\overline{A_i}, \overline{B_i}) \longleftarrow \mathsf{Add}((W_i, Z_i), \mathsf{Enc}_{\mathsf{pk}_i}(-\tau))$ output the ciphertexts $(\overline{A_1}, \overline{B_1}), \ldots, (\overline{A_n}, \overline{B_n})$	$\xleftarrow{\forall i = 1, \ldots, n : (W_i, Z_i)}$	

Fig. 7. $\mathsf{TransDec}$. Given a ciphertext (A, B), it computes n ciphertexts $(\overline{A_1}, \overline{B_1}), \ldots, (\overline{A_1}, \overline{B_1})$ of the same plaintext, encrypted under $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, respectively.

Data Retrieval. Each client P_i, $i = 1, \ldots, n$, can get the result of the computation by first retrieving (from $\mathcal{C}$) the encryption of $f(m_1, \ldots, m_n)$ under its public key pk_i that has been computed during the subprotocol $\mathsf{TransDec}$, and then decrypting this ciphertext by using its corresponding private key sk_i.

3.1 Correctness

We assume that all participants follow our protocol decriptions. Furthermore, we assume that the initial setup Init has been performed as decribed in Figure 2, and that all clients (wishing to participate) sent their encrypted private data to the server $\mathcal{C}$ as depicted in Figure 3. Under these assumptions, we show that the remaining protocols $\mathsf{KeyProd}, \mathsf{Add}, \mathsf{Mult}$ and $\mathsf{TransDec}$ produce the desired outputs correctly.

$\mathsf{KeyProd}$: Observe that for $i = 1, \ldots, n$, the values (C_i, D_i) are just blinded versions (using τ_i) of the original input ciphertexts (A_i, B_i), which are then decrypted (using the master secret MK) and re-encrypted under the product key $\mathsf{Prod.pk}$. The resulting values (W_i, Z_i) can then be transformed back to encryptions of the original underlying plaintexts by using the additively homomorphic property of the BCP scheme, and substracting the blinding value τ_i again.

Add: The correctness of this algorithm follows immediately from the additively homomorphic property and we refer to [15] for details.

Mult: We show that the output ciphertext $(\overline{A}, \overline{B})$ of algorithm Mult is indeed an encryption (under $\mathsf{Prod.pk}$) of the multiplication of the underlying plaintexts m, m' of the two input ciphertexts $(A, B), (A', B')$ (encrypted under $\mathsf{Prod.pk}$), respectively. Observe that the decryption of $(\overline{A}, \overline{B})$ under the secret key corresponding to $\mathsf{Prod.pk}$ (as mention before, if $\mathsf{Prod.pk} = \prod_{i=1}^{n} \mathsf{pk}_i$ the corresponding secret key is $\sum_{i=1}^{n} \mathsf{sk}_i$) yields $z \cdot z' + \sigma' m + \sigma m' + (-\sigma\sigma')$ where $z = m - \sigma$, $z' = m' - \sigma'$, and σ, σ' are random blinding values (cf. Figure 6). The expansion of this term gives $mm' - \sigma' m - \sigma m' + \sigma\sigma' + \sigma' m + \sigma m' + (-\sigma\sigma') = mm'$, which is the desired result.

$\mathsf{TransDec}$: Recall that this protocol takes an encryption (A, B) under $\mathsf{Prod.pk}$ of a message m as input and outputs ciphertexts $(\overline{A_i}, \overline{B_i})$, which are encryptions of the same message m but under the different public keys pk_i, for all $i = 1, \ldots, n$. In this protocol, essentially, $\mathcal{C}$ blinds the ciphertext (A, B) by adding a random message τ to the underlying plaintext m, which $\mathcal{S}$ decrypts and encrypts again under pk_i, for all $i = 1, \ldots, n$. It is obvious that once $\mathcal{C}$ subtracted τ again, the resulting ciphertext will be an encryption of m under pk_i, for all $i = 1, \ldots, n$.

Since KeyProd is independent of the actual function f that is to be computed, we see that once the underlying message $f(m_1, \ldots, m_n)$ has been computed in the encrypted domain (using Add and Mult), the correctness of TransDec yields that each client P_i can retrieve its dedicated encryption of $f(m_1, \ldots, m_n)$, which it can successfully decrypt by using its corresponding private key sk_i, $i = 1, \ldots, n$.

4 Security Analysis

The following security analysis considers the semi-honest model only, meaning that all parties follow the protocol description but try to gather information about other parties' inputs, intermediate results, or overall outputs just by looking at the protocol's transcripts. As usual, security in this model is proven in the "real-vs.-ideal" framework [31, Ch. 7]: there is an ideal model where all computations are performed via an additional trusted party and it is then shown that all adversarial behavior in the real model (where there is no trusted party) can be simulated in the ideal model. We deal with each subprotocol individually which is possible due to the Composition Theorem for the semi-honest model [31, Theorem 7.3.3]. Note that the security of all our protocols is essentially based on the well-known concept of "blinding" the plaintext: Given an encryption of a message, we use the additively homomorphic property of the cryptosystem to add a random message to it, which blinds the original plaintext.

Recall that before the actual computations (i.e., the cryptographic protocol) are performed between servers $\mathcal{C}$ and $\mathcal{S}$, there is only the initial setup of the BCP cryptosystem and the step where clients $P_1, \ldots, P_n$ store their encrypted private inputs $m_1, \ldots, m_n$ on the server $\mathcal{C}$ – for these two steps, the security follows from the semantic security of the BCP cryptosystem. Since we assume no collusion at all between any of the participating parties, it remains to show that neither $\mathcal{C}$ nor $\mathcal{S}$ learn anything from the cryptographic protocol (consisting of KeyProd, Add, Mult and TransDec) computing the n encryptions of $f(m_1, \ldots, m_n)$ under the clients' public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, respectively.

KeyProd: The only data sent is fresh ciphertexts. Due to the blinding values τ_i, $i = 1, \ldots, n$, and the semantic security of the BCP cryptosystem, these encryptions are indistinguishable from random ciphertexts and are therefore easily simulatable by encryption of, say, 1. Hence, both servers $\mathcal{C}$ and $\mathcal{S}$ do not learn anything at all from these ciphertexts.

Add: This algorithm is non-interactive and does not involve the server $\mathcal{S}$ at all, so we are only concerned about $\mathcal{C}$. For $\mathcal{C}$, however, no information leakage is assured by the semantic security of the BCP scheme since Add just uses the additively homomorphic property of the cryptosystem.

Mult: Again, the only data sent is fresh ciphertexts of blinded messages and due to the semantic security of the underlying cryptosystem, we can simulate these by random encryptions.

TransDec: Basically, the security argument here is the same as for the protocol KeyProd. The first step of TransDec is for $\mathcal{C}$ to blind the underlying message of the ciphertext (A, B) (which is encrypted under $\mathsf{Prod.pk}$) with the random message τ. This ensures that $\mathcal{S}$ does not learn any information about the original plaintext when receiving the "blinded" ciphertext (C, D). On the other hand, since $\mathcal{C}$ receives *fresh* encryptions under the public keys of the clients, he gets no information about the underlying plaintext whatsoever. In the language of simulations, a formal proof would amount to simulating the views which again is possible by using random encryptions due to the semantic security of the BCP scheme.

5 Variants

Recall that the main goal of our construction was to get rid of any interaction with the users. At the same time, our solution is very efficient compared to other existing work in similar scenarios. There are certain application scenarios, however, where the interaction with users is explicitly wanted, e.g., when the server is allowed to learn the result upon the approval of *all* participating users. In this section, we give a few variants of our original proposal that allow leveraging our efficient solution to such applications as well:

1. *Intermediate Key Aggregation.* If the (encrypted) private data from a client Q is not sent to the server $\mathcal{C}$ directly but goes through a chain of intermediate clients, this variant allows for the secure aggregation of intermediate public keys to Q's encrypted data and hence reduces work that otherwise needs to be done by the protocol $\mathsf{KeyProd}$.
2. *Disclosure by Clients' Approval.* This variant achieves a solution to the following scenario: Assume that clients are not supposed to see the result of $\mathcal{C}$'s computation. However, if *all* participating clients give their approval, $\mathcal{C}$ is able to read the result (while clients stay oblivious to this result).
3. *Interactive Decryption by all Clients.* If clients should learn the result only if *all* participating clients get together (and not each client independently as in our original construction), this variant can be run instead of protocol $\mathsf{TransDec}$ in order to make the decryption interactive between all clients. Decryption is successful if and only if all clients participate in this interaction (again in the semi-honest model).

Intermediate Key Aggregation. Assume that the (encrypted) private data of a client Q participating in our protocol is not sent to the server $\mathcal{C}$ directly (as depicted in Figure 3), but goes through other clients $Q_1, \ldots, Q_\ell$ (a subset of all clients $P_1, \ldots, P_n$). The protocol presented here optimizes $\mathsf{KeyProd}$ in this scenario: Recall, that $\mathsf{KeyProd}$ takes all the participating clients' ciphertexts (A_i, B_i) as input and transforms these to encryptions under the product $\mathsf{Prod.pk}$ of all participating public keys. The optimization we aim for does the aggregation of the public keys of the clients $Q, Q_1, \ldots, Q_\ell$ *before* the ciphertexts end up at the server $\mathcal{C}$. More precisely, let (A, B) be a ciphertext of a message m encrypted under a public key pk which arrives at client Q_i, $i = 1, \ldots, \ell$. This client can then use its own public key pk_i in order generate an encryption of the same message m but encrypted under the product $\mathsf{pk} \cdot \mathsf{pk}_i \mod N^2$. This "key aggregation" is one step that otherwise had to be done by the second server $\mathcal{S}$ in the original algorithm $\mathsf{KeyProd}$. Since Q_i knows its own private key, it can use the first component A and raise it to the power of its private key. The result of this can then be multiplied with the second component B which transforms the ciphertext (A, B) to an encryption under $\mathsf{pk} \cdot \mathsf{pk}_i \mod N^2$. Details of this are described in Figure 8.

```
**Client** $Q_i$ has private key $\mathsf{sk}_i$

Upon retrieval of ciphertext $(A, B)$ (encrypted under $\mathsf{pk}$):
  $\overline{B} \longleftarrow A^{\mathsf{sk}_i} \cdot B \mod N^2$
output ciphertext $(A, \overline{B})$
```

Fig. 8. Upon retrieval of a ciphertext (A, B) of m under pk, client Q_i produces an encryption of m under $\mathsf{pk} \cdot \mathsf{pk}_i \mod N^2$, $i = 1, \ldots, \ell$.

The correctness of this algorithm is immediately seen, since when $(A, B) = (g^r \mod N^2, \mathsf{pk}^r(1 + mN) \mod N^2)$, we have that $\overline{B} = A^{\mathsf{sk}_i + \mathsf{sk}}(1 + mN) \mod N^2$ since $\mathsf{pk}^r = g^{r\mathsf{sk}} = A^{\mathsf{sk}} \mod N^2$.

The security of this variant is implied by the security of the BCP cryptosystem. This is because the first client Q in the chain of clients is simply giving away a fresh encryption under its own public key.

Disclosure by Clients' Approval. Assume that the clients are not allowed to see the actual result $f(m_1, \ldots, m_n)$ of the computation done by the server $\mathcal{C}$, but if *all* clients approve it, $\mathcal{C}$ should be able to retrieve the result (while all clients stay oblivious). More precisely, let (A, B) denote the encrypted result of the computation done by the server $\mathcal{C}$ before applying algorithm TransDec to it (so (A, B) is an encryption under the public key Prod.pk). We assume that the participating clients $P_1, \ldots, P_n$ are not supposed to see the (encrypted) result, but on their approval (from *all* of them), the server $\mathcal{C}$ should be able to decrypt (A, B) in order to see the result of the computation. To achieve this, we can run protocol ODApproval of Figure 9 *instead* of TransDec. The basic idea of this protocol is to run the "key aggregation" in reversed order, meaning that $\mathcal{C}$ blinds the first component A of the encrypted result and sends it to the clients. By using their respective private keys, each client returns a modified version of A. Now, recall that the result is encrypted under the product Prod.pk of all clients' public keys. Therefore, these modified versions of A can be used by $\mathcal{C}$ to "divide out" the public keys of each

Clients $P_1, \ldots, P_n$ with secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$		**Server** $\mathcal{C}$, stores Prod.pk and $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$
		On input ciphertext (A, B) (encrypted under Prod.pk): Pick random $\rho \overset{U}{\longleftarrow} \mathbb{Z}_N$ $(\overline{A}, \overline{B}) \longleftarrow (A \cdot g^{\rho} \mod N^2, B \cdot \mathsf{Prod.pk}^{\rho} \mod N^2)$
	$\xleftarrow{\overline{A} \text{ to } \textit{all} \text{ clients}}$	
Client P_i $(i = 1, \ldots, n)$: $X_i \longleftarrow \overline{A}^{\mathsf{sk}_i} \mod N^2$	$\xrightarrow{X_i}$	
		Upon retrieval of $X_1, \ldots, X_n$: $K \longleftarrow \prod_{i=1}^{n} X_i \mod N^2$ $m \longleftarrow \frac{\overline{B}/K - 1 \mod N^2}{N}$ Output plaintext m

Fig. 9. ODApproval: Server $\mathcal{C}$ asks all participating clients for their approval to disclose the encrypted result to $\mathcal{C}$ while all clients stay oblivious to this result.

individual client seperately, ending up with a encryption under the public key 1 which simply is the plaintext itself.

The correctness of this protocol is shown by proving that if (A, B) is an encryption of m' under the public key $\mathsf{Prod.pk}$, then the output m of $\mathsf{ODApproval}$ equals this message m'. If (A, B) was encrypted by using randomness r, then

$$(\overline{A}, \overline{B}) = (g^{\tau} \mod N^2, \mathsf{Prod.pk}^{\tau}(1 + m'N) \mod N^2), \tag{9}$$

where $\tau = r + \rho$. But $K = \prod_{i=1}^{n} \overline{A}^{\mathsf{sk}_i} \mod N^2 = \mathsf{Prod.pk}^{\tau} \mod N^2$ and so

$$m = \frac{(1 + m'N) - 1 \mod N^2}{N} = m'. \tag{10}$$

As for the security, we note that the clients do not learn anything at all since the plaintext is encoded in the second component of the ciphertext (A, B) which is never sent to any of the clients. So the plaintext remains information theoretically secure.

Concerning the server $\mathcal{C}$, we recall that due to the semantic security of the BCP cryptosystem, $\mathcal{C}$ is not able to compute the randomness used to encrypt (A, B) and so the clients' messages X_i are indistinguishable from random elements in $\langle g \rangle$. This also implies that as long as one of the messages X_i is missing, $\mathcal{C}$ is not able to decrypt: Assume that the final message X_n is missing and $\mathcal{C}$ already computed $K' = \prod_{i=1}^{n-1} X_i \mod N^2$. Trying to decrypt $(\overline{A}, \overline{B}) = (g^r \mod N^2, \mathsf{Prod.pk}^r(1 + mN) \mod N^2)$ using K' results in the ciphertext $(g^r \mod N^2, \mathsf{pk}_n^r(1 + mN) \mod N^2)$, i.e., an encryption under the public key of P_n which is the client from whom the message X_n is still missing.

Interactive Decryption by all Clients. Assume that we want all participating clients to decrypt the result of $\mathcal{C}$'s computation together (in an interactive protocol) so that neither the server $\mathcal{C}$ nor the server $\mathcal{S}$ learn the result, but all clients do (if and only if all clients participate). Essentially, this can be achieved by using the protocol $\mathsf{ODApproval}$ (cf. Figure 9). Server $\mathcal{C}$ sends the re-randomized encryption $(\overline{A}, \overline{B})$ to all clients and then, instead of sending the values $X_1, \ldots, X_n$ to the server $\mathcal{C}$, for each $i = 1, \ldots, n$, client P_i broadcasts its respective value X_i to all other clients. Upon retrieval of all these values $X_1, \ldots, X_n$, each client is now able to decrypt (in the same way as $\mathcal{C}$ does in Figure 9) the ciphertext $(\overline{A}, \overline{B})$ to reveal the underlying result of $\mathcal{C}$'s computation. The correctness of this protocol follows from the correctness of $\mathsf{ODApproval}$.

The same argument as for the previous variant shows that all messages $X_1, \ldots, X_n$ need to be received in order to decrypt the ciphertext $(\overline{A}, \overline{B})$. So decryption is not possible until all clients broadcasted their respective value X_i. Since the clients never see the (encrypted) private inputs of other clients, they cannot learn more than the decryption of $(\overline{A}, \overline{B})$ which is the result of the computation done by the server $\mathcal{C}$.

6 Performance

The performance of our contribution depends on the security parameter κ, the number of clients n, the number of additions and multiplications performed, and the network performance (bandwidth and latency). All algorithms, except Init are solely based on arithmetic operations (addition, subtraction, multiplication, exponentiation, inversion, division, comparison) modulo N or N^2 and random number generation. The size of N grows linearly with κ. A full overview of the complexity of each protocol step is given in Figure 10 (a). To show that our system can be used in practice, we implemented a proof-of-concept version of all protocols in python. Because the speed of the implementation depends mainly on the speed of the bignum library, we used the GMP library though pythons *gmpy* module. The GMP library is known to be asymptotically faster than pythons native bignum library, when it comes to processing bigger numbers. GMP offers all basic operations we need, except for the generation of safe primes numbers. Therefore, we also used the OpenSSL library through a custom C-binding, but solely for generating safe prime numbers. We used a TCP network connection over the loopback interface for transferring data between the two servers.

The runtime of our code is heavily influenced by the size of N. We recommend a size of N of at least 1024 bit, while 1536 or 2048 bit are better choices for security reasons. Therefore, we performed all benchmarks with these three security parameters. We performed all tests on a *Lenovo Thinkpad T410s* with an *Intel Core i5 M560* running at 2.67 GHz with *Debian Sid*, *Python 2.7.3rc2* and *gmpy 1.15-1*. We ran all clients and both servers on the same host, so that network latency can be ignored. Algorithm Init has the worst time complexity due to the generation of two safe primes. Its runtime is expected to vary, because the number of instructions it needs to execute in order to find two suitable primes depends on the random numbers generated by the algorithm. Figure 10 (b) shows the runtime distribution of Init, depending on κ. 20 tests were performed for each choice of κ, and it is clearly visible that the runtime varies a lot. In the next step, we determined how long it takes to encrypt all client's inputs, transcode them at the server and hand the result back to the clients (omitting the initialization step). Figure 10 (c) shows that our implementation scales linearly with the number of clients. Without any arithmetic operations, we can perform a full protocol run with 16 clients in 1.7 seconds, using a 1536 bit modulus. Finally, we were interested in the runtime of Add and Mult. Figure 10 (d) shows that an addition is much faster than a multiplication. With a 1536 bit modulus, about 25,000 additions, but only 5 multiplications can be performed per second. The results show that our scheme is really useable in practice.

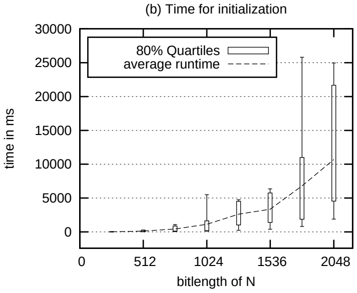
7 Applications

Finally, we present two applications of our general-purpose construction, namely privacy-preserving face recognition and private smart metering, which benefit from the non-interactive nature of our protocols. In order to demonstrate the practicability of our approach, we decided to implement the much more complex application of these two, namely privacy-preserving face recognition.

Privacy-Preserving Face Recognition. Face recognition is a widely-used tool in many areas of modern everyday life, among which social networks probably is one of the most important. Many social networks use face recognition tools for things like automatic photo tagging [5] or in order to help law enforcement agencies to prosecute suspected persons [3] (to name just a few). This is usually being done without the explicit consent of users, raising important privacy concerns, as convincingly demonstrated in [23] and [52]. Therefore, Erkin et al. [23] proposed the first privacy-preserving protocol

(a) Complexity of the protocol

Algorithm	Time	Traffic in bits	Round trips
Init	$O(\kappa^5/\log(\kappa)^2)$ on $\mathcal{S}$	4κ	0
Data Upload	$O(\kappa^3)$ on each client $O(n\kappa^2)$ on $\mathcal{C}$	$6n\kappa$	0.5
KeyEval	$O(n\kappa^3)$ on $\mathcal{S}$ $O(n\kappa^2)$ on $\mathcal{C}$	8κ	1
Add	$O(\kappa^2)$ on $\mathcal{C}$	0	0
Mult	$O(\kappa^3)$ on $\mathcal{C}$ $O(\kappa^3)$ on $\mathcal{S}$	12κ	1
TransDec	$O(n\kappa^3)$ on $\mathcal{C}$ $O(n\kappa^3)$ on $\mathcal{S}$	$4(n+1)\kappa$	1
Data Retrieval	$O(\kappa^3)$ on each client	$4n\kappa$	0.5



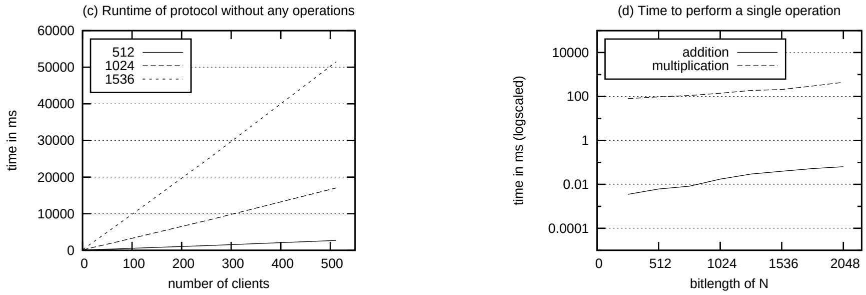


Fig. 10. Complexity and runtime analysis of the protocol.

for face recognition which, however, heavily relies on user interaction. Unfortunately, their solution generally does not work in the social network setting where users access their profiles by using resource-constrained mobile devices which are not always online. Therefore, having a completely non-interactive (with the users) solution is crucial in such scenarios. Additionally, users typically want to use their own public keys to encrypt their private images (which, again, is not possible in previous solutions).

Similarly to Erkin et al. [23], the database of "known images" (i.e., when given a face image, we want to find out whether it is in the database or not) is unencrypted in our setting. For instance, in the scenario of helping with the prosecution of suspected persons, the database of known images consists of the suspected persons and is therefore known by the social network provider in plaintext (unencrypted). On the other hand, it is desirable to have all users' profile pictures (or images of users' holidays etc.) encrypted, so that the social network provider is only able to recognize faces that are found in the database while all other faces remain hidden.[2] To achieve this goal, we adapt the privacy-preserving protocol by Erkin et al. [23] to our framework and hence rely on the standard eigenfaces recognition algorithm [54, 53], which can be summarized as follows:

Enrolment. This is the first phase, in which a "face space" is determined on the basis of a set of M training images $\Theta_1, \ldots, \Theta_M$ (represented as a vector of length L). Later on, in the recognition phase,

[2] Note that the correct recognition is only assured with a certain probability, since the tool of face recognition is error-prone. In the implementation that we use (cf. [23]), the correct classification rate is approximately 96%.

face images will be projected onto this "face space" and matched against the training images. Creating the "face space" is done by

1. computing the "average face" $\mathit{\Psi} = \frac{1}{M}\sum_{i=1}^{M} \mathit{\Theta}_i$,
2. computing the "difference vectors" $\mathit{\Phi}_i = \mathit{\Theta}_i - \mathit{\Psi}$ for all $i = 1, \ldots, M$,
3. applying the Principal Component Analysis (PCA) to the covariance matrix $C = \frac{1}{M}AA^T$ where A is the matrix $[\mathit{\Theta}_1\ \mathit{\Theta}_2\ \ldots\ \mathit{\Theta}_M]$ (see [54] for details).[3] This yields orthonormal eigenvectors with associated eigenvalues of C,
4. selecting $K \ll M$ eigenvectors $u_1, \ldots, u_K$ (the "eigenfaces") associated to the K largest eigenvalues,
5. and projecting $\mathit{\Theta}_1, \ldots, \mathit{\Theta}_M$ to the "face space" spanned by $u_1, \ldots, u_K$ to get their "feature vectors" $\mathit{\Omega}_1, \ldots, \mathit{\Omega}_M$ where $\mathit{\Omega}_i = (\omega_{i1}, \ldots, \omega_{iK})^T$ with $\omega_{ij} = u_j^T(\mathit{\Theta}_i - \mathit{\Psi})$ for all $j = 1, \ldots, K$ and $i = 1, \ldots, M$.

Recognition. Given a face image $\mathit{\Gamma}$, recognition is done by

1. ("Projection" step) projecting $\mathit{\Gamma}$ onto the face space by computing its "feature vector" $V = (v_1, \ldots, v_K)^T$ with $v_j = u_j^T(\mathit{\Gamma} - \mathit{\Psi})$ for all $j = 1, \ldots, K$,
2. ("Distance" step) computing the squares $D_1, \ldots, D_M$ of the Euclidean distances between V and $\mathit{\Omega}_1, \ldots, \mathit{\Omega}_M$ as $D_j = \|V - \mathit{\Omega}_j\|^2$ for $j = 1, \ldots, M$,
3. ("Minimum" step) and reporting a match if the smallest distance $D_{\min} = \min\{D_1, \ldots, D_M\}$ is smaller than a given threshold τ.

We stress that since the database of "known images" is known by the social network provider, all steps of the enrolment phase can be done in the clear, without using any form of encryption. Hence, we only have to transform the steps of the recognition phase into the encrypted domain. We do this by a rather straight-forward adaption of the protocols in [23] to our framework:

Encrypting Images. Assume that a user U wants to upload an encryption of a private face image $\mathit{\Gamma}$ under its respective public key pk to the server $\mathcal{C}$ (here, $\mathcal{C}$ is the social network provider). Recall that we represent images as vectors of length L, i.e., $\mathit{\Gamma} = (\mathit{\Gamma}_1, \ldots, \mathit{\Gamma}_L)^T$. User U encrypts the image $\mathit{\Gamma}$ component-wise, which we denote by $\mathsf{EncImg}_{\mathsf{pk}}(\mathit{\Gamma})$, i.e., $\mathsf{EncImg}_{\mathsf{pk}}(\mathit{\Gamma}) = (\mathsf{Enc}_{\mathsf{pk}}(\mathit{\Gamma}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(\mathit{\Gamma}_L))^T$.

Projection. Given an image encryption $\mathsf{EncImg}_{\mathsf{pk}}(\mathit{\Gamma}) = (c_1, \ldots, c_L)^T$ of a face image $\mathit{\Gamma}$ under a user's public key pk, server $\mathcal{C}$ computes for all $i = 1, \ldots, K$:

$$\overline{v_i} := \mathsf{Enc}_{\mathsf{pk}}\left(-\sum_{j=1}^{L} u_{ij}\mathit{\Psi}_j\right) \cdot \prod_{j=1}^{L} c_j^{u_{ij}} \mod N^2,$$

where u_{ij} denotes the j-th component of the i-th eigenface u_i and $\mathit{\Psi}_j$ denotes the j-th component of the average face $\mathit{\Psi}$. This way, $\mathcal{C}$ obtains the encrypted feature vector $\overline{V} = (\overline{v_1}, \ldots, \overline{v_K})^T$ corresponding to $\mathit{\Gamma}$.

Distance. Given the projected face vector $\overline{V} = (\overline{v_1}, \ldots, \overline{v_K})^T$ from the "projection" step, $\mathcal{C}$ computes the Euclidean distances to the feature vectors $\mathit{\Omega}_i = (\omega_{i1}, \ldots, \omega_{iK})^T$ for $i = 1, \ldots, M$, with the help

[3] Actually, PCA is applied to the much smaller matrix A^TA with appropriate post-processing [54].

of server $\mathcal{S}$ as follows: First, $\mathcal{C}$ computes for all $i = 1, \ldots, M$: $\sigma_{i1} := \mathsf{Enc}_{\mathsf{pk}}\left(\sum_{j=1}^{K} \omega_{ij}^2\right)$ and $\sigma_{i2} := \prod_{j=1}^{K} \overline{v_j}^{(-2\omega_{ij})}$. Note that $\mathcal{C}$ can perform these computations without interacting with $\mathcal{S}$. Second, $\mathcal{C}$ computes $\sigma_3 := \prod_{i=1}^{M} \mathsf{Mult}(\overline{v_i}, \overline{v_i})$. Note that Mult runs interactively with server $\mathcal{S}$. Finally, for all $i = 1, \ldots, M$, $\mathcal{C}$ computes the results $\delta_i := \sigma_{i1} \cdot \sigma_{i2} \cdot \sigma_3$, which are encryptions of the squares of the Euclidean distances of V with $\Omega_1, \ldots, \Omega_M$.

Minimum. In this final step, $\mathcal{C}$ performs a joint computation with $\mathcal{S}$ in order to determine the minimum of the encrypted distances $\delta_1, \ldots, \delta_M$ together with an encryption of its index $\mathsf{id} \in \{1, \ldots, M\}$ and to check whether this minimum is smaller than a given threshold τ. This is done by encrypting τ (under pk) and treating it as an additional (encrypted) distance with the special index 0 (which we encrypt under pk as well). The distances as well as the indices have to be stored encrypted since only the user U should be able to see the final result of this minimum-finding. The protocol to perform this step in a privacy-preserving manner is exactly as in [23] and we refer the reader to this reference for details since the protocol is rather complex and would go beyond the scope of this paper. We stress that one simply has to replace the parties Alice and Bob in [23] by servers $\mathcal{S}$ and $\mathcal{C}$, respectively, and the Paillier cryptosystem [49] by the BCP cryptosystem [15]. Also, we note that in our implementation we solely used the BCP cryptosystem, while [23] switches to the homomorphic encryption scheme DGK by Damgård, Geisler, and Krøigaard [20] for efficiency reasons.

We stress that we deliberately implemented all protocols only by using our general-purpose construction as is, *without* any tricks to optimize the performance, such as switching to the much more efficient DGK cryptosystem for computing the minimal distance, or treating the fact that most computations can be done in an offline pre-computation phase. Note, however, that all such tricks can be applied to our protocols in exactly the same way as it is done in [23]. We did not implement these tricks in order to show the performance of our general-purpose construction in its plain as-is state without tweaking it to specific application scenarios.

Performance Analysis. We implemented the complete protocol for privacy-preserving face recognition, as described above, in python while basing it on our implementation of our general-purpose construction described in Section 6. This means that we used the GMP and OpenSSL libraries, while all tests were performed on a *Lenovo Thinkpad T410s* with an *Intel Core i5 M560* running at 2.67 GHz, running *Debian Sid* with *Python 2.7.3rc2* and *gmpy 1.15-1*. The security parameter of our general-purpose construction was fixed to $\kappa = 1024$, i.e., the size of the RSA-modulus N is 1024 bits.

In order to avoid confusion, we used the same set of parameters for the privacy-preserving face recognition protocol as [23]. This also ensures the same reliability results as in [23] (achieving a correct classification rate of approximately 96%). In particular, we used the "ORL Database of Faces" from AT&T Laboratories Cambridge [1] containing 10 images of 40 different subjects, yielding a total amount of 400 images.[4] These images are of size 92 × 112 pixels, which we represented as vectors of length $L = 92 \cdot 112 = 10304$. It is demonstrated in [23] that $K > 12$ eigenfaces in the enrolment phase do not significantly increase the correct classification rate, which is why we used their suggestion of $K = 12$ (for $M = 10$, we used $K = 5$ instead). Table 1 depicts the results of our performance analysis

[4] We actually use a subset of these which is determined through the size M of the database of "known images".

		Traffic	Traffic	Runtime w/
Database	Runtime	$\mathcal{C}$ to $\mathcal{S}$	$\mathcal{S}$ to $\mathcal{C}$	remote $\mathcal{S}$
size M	in sec.	in MB	in MB	in sec.
10	106	0.6	0.7	100
50	297	4	4.2	294
100	533	8	8.5	506
150	761	12	13	744
200	1004	16	17	970

Table 1. Complexity of performing privacy-preserving face recognition with our general-purpose construction.

of the complete privacy-preserving face recognition protocol (containing both $\mathcal{C}$'s and $\mathcal{S}$'s costs). The first column shows the amount M of feature vectors stored in the database, while the second column shows the runtime (wall clock time in seconds) of the full protocol per invocation with one face image (that is to be matched with the database). The runtime shown in the second column contains the enrolment phase which in each case took less than 1% time of the shown overall runtime. We therefore included the enrolment phase here, since it does not significantly change the overall performance. The communication complexity (measured with iptables) is summarized in the other two columns: the third column shows the amount of data (in megabytes) sent from server $\mathcal{C}$ to server $\mathcal{S}$, while the fourth column shows the total traffic from $\mathcal{S}$ back to $\mathcal{C}$.

Finally, the fifth column shows the runtime (in seconds) of the full protocol (per single query) when using a remote server $\mathcal{S}$ over the Internet. For this remote setup, we connected our Lenovo Thinkpad T410s (server $\mathcal{C}$) with a 20 Mbit/s consumers cable connection to the Internet and started our server $\mathcal{S}$ implementation on a remote system equipped with a 3.2 GHz *AMD Phenom II X6 1090T* processor in a datacenter connected to the Internet with a Gigabit Ethernet adapter. The average round trip time to that system was 32 ms, measured with the linux "ping" utility. Observe that the overall runtime when running server $\mathcal{S}$ remotely is less than when running $\mathcal{S}$ locally. This is due to the fact that the remote system that we used has higher performance than the local one.

In summary, for a database of size $M = 200$, a full protocol run requires approximately 16 minutes (both when running $\mathcal{S}$ locally or remotely) for checking whether a given encrypted image is contained in the database or not. This demonstrates that our general-purpose construction can really be used in practice for performing facial recognition in a privacy-preserving manner in the social network setting, even *without* any efficiency optimizations. Again, we stress that we can apply the same performance tweaks to our protocol as Erkin et al. [23], which would yield a significant efficiency boost. Also, we note that we did not use explicit parallelism in our implementation, so that the performance can moreover be improved by using multiple CPU cores. We leave the elaboration of such possibilities as interesting future work.

Private Smart Metering. Another interesting application domain, actively promoted by many governments, concerns the deployment of smart grids for modernizing the distribution networks of electricity, gas or water. For such services, smart meters record the consumption of individual consumers on a fine-grained basis and send all collected data to a central authority (the *supplier*), which in turn uses these inputs to compute overall consumptions, bills (by using dynamic pricing schemes), usage patterns, or to detect fraud or leakage in other utilities. Due to the collection of massive amount of sensitive data, privacy concerns have been raised in the past years, and various solutions protecting the consumers' privacy have been proposed for different computing tasks of the supplier. Current solutions either only consider eavesdropping outsiders (and not the supplier) [43], require interaction and high computational overhead at the smart meters [28, 42] or rely on trusted components alongside each smart meter [47, 38].

Our construction of Section 3 offers a non-interactive protocol with very low computational costs at the smart meters. The smart meters would act as the clients in our protocol description of Section 3, sending their encrypted private data to the supplier $\mathcal{C}$. With our cryptographic protocol between $\mathcal{C}$ and a second server $\mathcal{S}$, the supplier $\mathcal{C}$ can essentially compute any function on the consumers' inputs in a privacy-preserving way. Furthermore, concerning private information aggregation, we can extend the distributed incremental data aggregation approach of [43] in order to protect the consumers' privacy from the supplier $\mathcal{C}$ as well. In [43], the idea is to encrypt each consumer's private data with an additively homomorphic encryption scheme under the public key of the supplier $\mathcal{C}$ and then send this encrypted information through other households in the neighbourhood (this is due to the use of short-range communication networks) in order to finally reach the supplier. Intermediate households aggregate their private data to the one they receive by using the additive property of the cryptosystem. Once the supplier received the data from all these chains of households, it aggregates these encryptions together to get the aggregation of all consumers' inputs. In [43], only eavesdropping adversaries (such as the intermediate households) are considered. By employing our construction variant "Intermediate Key Aggregation", we can protect the consumers' privacy from intermediate households as well as from the supplier: Consumer P_1 encrypts its private data by using its own public key pk_1 and sends it to the next consumer P_2 which in turn uses the "Intermediate Key Aggregation" algorithm to transform the encryption into an encryption under the product $\mathsf{pk}_1\mathsf{pk}_2 \mod N^2$ of the two consumers' public keys. Furthermore, P_2 encrypts its own private input under this product $\mathsf{pk}_1\mathsf{pk}_2 \mod N^2$ as well and uses the additively homomorphic property of the BCP cryptosystem to aggregate its encrypted input to the encrypted input of the first consumer P_1. These steps continue through the whole chain of consumers until the supplier $\mathcal{C}$ is reached. $\mathcal{C}$ can now aggregate all the remaining consumers' encrypted inputs (similar to [43]) but still is unable to see any of the underlying plaintexts. Depending on the application, we can continue in three ways:

1. $\mathcal{C}$ decrypts the result jointly with the second server $\mathcal{S}$ (recall that $\mathcal{S}$ still has the master secret).
2. We use the variant "Disclosure by Clients' Approval" which means that $\mathcal{C}$ can only decrypt if all consumers approve it.
3. $\mathcal{C}$ sends the encrypted result of the aggregation (or any other evaluation on the basis of this aggregation) back to the consumers for local decryption.

We stress that due to the high efficiency of our solution, it is particularly interesting in the smart grid scenario as it minimizes computational overhead at the smart meters.

8 Conclusion

Assuming the existence of two non-colluding but untrusted servers, we presented an efficient general-purpose SMC protocol that requires no interaction of the users at all, and allows the evaluation of arbitrary functions on inputs that are encrypted under different independent public keys. We showed our protocol to be secure in the semi-honest model and highlighted its practicability by giving experimental results. Two application scenarios, one on privacy-preserving face recognition and one on private smart metering, underlined the applicability of our construction to the real-world.

We consider the extension to the malicious adversarial model as future work. Furthermore, we are planning to experiment with certain functionalities used in other application scenario, e.g., computations on medical health records. To this end, it seems useful to integrate our work into tools for automating SMC [8, 34].

References

1. The database of faces. AT&T Laboratories Cambridge, http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
2. Google admits weekend privacy breach. Fox News (March 10, 2009), http://www.foxnews.com/story/0,2933,508324,00.html
3. Guess what? facebook monitors postings and chats. Daily News (July 16, 2012), http://articles.nydailynews.com/2012-07-16/news/32701753_1_facebook-employees-conversations-software
4. The facebook privacy scandal. ABC News (October 19, 2010), http://abcnews.go.com/GMA/Consumer/facebook-privacy-scandal-facebooks-watergate/story?id=11912201
5. Camurati: Recognizing the dangers behind facial recognition. Newsday (July 27, 2012), http://www.newsday.com/opinion/viewsday-1.3683911/camurati-recognizing-the-dangers-behind-facial-recognition-1.3865294
6. A face is exposed for AOL searcher no. 4417749. The New York Times (August 9, 2006), http://www.nytimes.com/2006/08/09/technology/09aol.html
7. Armknecht, F., Katzenbeisser, S., Peter, A.: Group homomorphic encryption: Characterizations, impossibility results, and applications. Designs, Codes and Cryptography DOI: 10.1007/s10623-011-9601-2
8. Ben-David, A., Nisan, N., Pinkas, B.: Fairplaymp: a system for secure multi-party computation. In: ACM Conference on Computer and Communications Security. pp. 257–266. ACM (2008)
9. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT. LNCS, vol. 6632, pp. 169–188. Springer (2011)
10. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: ESORICS. LNCS, vol. 5283, pp. 192–206. Springer (2008)
11. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T.P., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Financial Cryptography. LNCS, vol. 5628, pp. 325–343. Springer (2009)
12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: ITCS. pp. 309–325. ACM (2012)
13. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. In: FOCS. pp. 97–106. IEEE (2011)
14. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: CRYPTO. LNCS, vol. 6841, pp. 505–524. Springer (2011)

15. Bresson, E., Catalano, D., Pointcheval, D.: A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: ASIACRYPT. LNCS, vol. 2894, pp. 37–54. Springer (2003)
16. Catrina, O., Kerschbaum, F.: Fostering the uptake of secure multiparty computation in e-commerce. In: ARES. pp. 693–700. IEEE Computer Society (2008)
17. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC. pp. 11–19. ACM (1988)
18. Choi, S.G., Elbaz, A., Juels, A., Malkin, T., Yung, M.: Two-party computing with encrypted data. In: ASIACRYPT. LNCS, vol. 4833, pp. 298–314. Springer (2007)
19. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: EUROCRYPT. LNCS, vol. 2045, pp. 280–299 (2001)
20. Damgård, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for on-line auctions. In: ACISP. LNCS, vol. 4586, pp. 416–430. Springer (2007)
21. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: CRYPTO. LNCS, vol. 3621, pp. 378–394 (2005)
22. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO. LNCS, vol. 7417, pp. 643–662. Springer (2012)
23. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: PET. pp. 235–253. LNCS 5672 (2009)
24. Erkin, Z., Veugen, T., Toft, T., Lagendijk, R.: Generating private recommendations efficiently using homomorphic encryption and data packing. IEEE Transactions on Information Forensics and Security (to appear) (2012)
25. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: STOC. pp. 554–563. ACM (1994)
26. Galindo, D., Herranz, J.: On the security of public key cryptosystems with a double decryption mechanism. Inf. Process. Lett. 108(5), 279–283 (2008)
27. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
28. Garcia, F.D., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: STM. LNCS, vol. 6710, pp. 226–238. Springer (2010)
29. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis (2009)
30. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: EUROCRYPT. LNCS, vol. 7237, pp. 465–482. Springer (2012)
31. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
32. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. pp. 218–229. ACM (1987)
33. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: CRYPTO. pp. 132–150. LNCS 6841 (2011)
34. Henecka, W., Kögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: Tasty: tool for automating secure two-party computations. In: ACM Conference on Computer and Communications Security. pp. 451–462. ACM (2010)
35. Ishai, Y., Kushilevitz, E.: Private simultaneous messages protocols with applications. In: ISTCS. pp. 174–184 (1997)
36. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: TCC. LNCS, vol. 5444, pp. 294–314. Springer (2009)

37. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: EUROCRYPT. LNCS, vol. 4515, pp. 97–114. Springer (2007)
38. Jawurek, M., Johns, M., Kerschbaum, F.: Plug-in privacy for smart metering billing. In: PETS. LNCS, vol. 6794, pp. 192–210. Springer (2011)
39. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. IACR Cryptology ePrint Archive 2011, 272 (2011)
40. Kamara, S., Mohassel, P., Riva, B.: Salus: a system for server-aided secure function evaluation. In: ACM Conference on Computer and Communications Security. pp. 797–808. ACM (2012)
41. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design. IACR Cryptology ePrint Archive 2010, 79 (2010)
42. Kursawe, K., Danezis, G., Kohlweiss, M.: Privacy-friendly aggregation for the smart-grid. In: PETS. LNCS, vol. 6794, pp. 175–191. Springer (2011)
43. Li, F., Luo, B., Liu, P.: Secure and privacy-preserving information aggregation for smart grids. IJSN 6(1), 28–39 (2011)
44. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: EUROCRYPT. LNCS, vol. 4515, pp. 52–78. Springer (2007)
45. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC. ACM (2012)
46. Lu, S., Ostrovsky, R.: Distributed oblivious ram for secure two-party computation. Cryptology ePrint Archive, Report 2011/384 (2011), `http://eprint.iacr.org/`
47. Molina-Markham, A., Shenoy, P., Fu, K., Cecchet, E., Irwin, D.: Private memoirs of a smart meter. In: BuildSys (2010)
48. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM Conference on Electronic Commerce. pp. 129–139 (1999)
49. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT. LNCS, vol. 1592, pp. 223–238. Springer (1999)
50. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: ASIACRYPT. LNCS, vol. 5912, pp. 250–267. Springer (2009)
51. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: ACM CCS. pp. 199–212. ACM (2009)
52. Sadeghi, A.R., Schneider, T., Wehrenberg, I.: Efficient privacy-preserving face recognition. In: ICISC. LNCS, vol. 5984, pp. 229–244. Springer (2009)
53. Turk, M.A., Pentland, A.P.: Face recognition using eigenfaces. In: CVPR. pp. 586–591. IEEE Comput. Sco. Press (1991)
54. Turk, M., Pentland, A.: Eigenfaces for recognition. J. Cognitive Neuroscience 3(1), 71–86 (Jan 1991)
55. Van Dijk, M., Juels, A.: On the impossibility of cryptography alone for privacy-preserving cloud computing. In: HotSec'10. pp. 1–8. USENIX (2010)
56. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. pp. 160–164. IEEE Computer Society (1982)