Efficiently Outsourcing Multiparty Computation Under Multiple Keys

Andreas Peter, Erik Tews, and Stefan Katzenbeisser, *Senior Member, IEEE*

***Abstract*— Secure multiparty computation enables a set of users to evaluate certain functionalities on their respective inputs while keeping these inputs encrypted throughout the computation. In many applications, 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 public key—drawbacks making the employment in practice very limited. We propose a novel technique based on additively homomorphic encryption that avoids all these drawbacks. This method is efficient, requires no user interaction whatsoever (except for data upload and download), and allows evaluating any dynamically chosen function on inputs encrypted under different 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 proven to be secure in the semi-honest model. By developing application-tailored variants of our approach, we demonstrate its versatility and apply it in two real-world scenarios from different domains, privacy-preserving face recognition and private smart metering. We also give a proof-of-concept implementation to highlight its feasibility.**

***Index Terms*— Information security, homomorphic encryption, secure multiparty computation, secure outsourcing.**

I. 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 [1]–[3].

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 [4], [5]. 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, a 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 $P_1, \ldots, P_n$ (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.

This setting has been addressed in many works before, while only providing partial solutions. Thus, we would like to recall related work and point out some limitations of possible approaches to solve our scenario, before explaining our approach.

Manuscript received August 1, 2013; revised October 23, 2013; accepted October 23, 2013. Date of publication November 1, 2013; date of current version November 11, 2013. The work of A. Peter was supported by the THeCS project as part of the Dutch national program COMMIT. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mauro Barni.

A. Peter is with the Services, Cybersecurity and Safety Group, Centre for Telematics and Information Technology, University of Twente, Enschede 7500 AE, The Netherlands (e-mail: a.peter@utwente.nl).

E. Tews and S. Katzenbeisser are with the Security Engineering Group, Department of Computer Science, Technische Universität Darmstadt/CASED, Darmstadt 64289, Germany (e-mail: e_tews@seceng.informatik.tu-darmstadt.de; katzenbeisser@seceng.informatik.tu-darmstadt.de).

Digital Object Identifier 10.1109/TIFS.2013.2288131

A. Previous Work

Most papers on SMC protocols are concerned with *interactive* solutions where all parties are actively involved in computing an arbitrary function on their respective inputs in a privacy-preserving manner [6]–[10]. 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]–[16], but again with interaction of the clients during the computations. Furthermore, Choi et al. [17] give a solution with minimal interaction of the clients, while relying on two non-colluding servers. 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. [18] 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 [19], 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 [20], 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. [21] 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 public keys, having similar efficiency shortcomings as all the other FHE schemes. More importantly, [21, Theorem B.1] proves (in the semi-honest model, similar to [22]) any solution that runs non-interactively with the clients in the *single* server setting to be impossible to realize (drawing on the impossibility of program obfuscation).

B. Approaches to Related Problems

Following the impossibility result by López-Alt et al. [21], we need at least two non-colluding servers, if we want complete non-interaction with the clients. Hence, we assume the existence of *two semi-honest, non-colluding but otherwise untrusted servers $\mathcal{C}$ and $\mathcal{S}$*. This assumption is very common both in the theoretical (e.g., [12], [17]) and the practical community (e.g., [11], [23], [24]). Next, we would like to point out two interesting approaches that solve a different but related problem:

1) **Single-Key Approach.** The second server $\mathcal{S}$ possesses a public-private key pair of some additively homomorphic encryption scheme. Every client encrypts its data using $\mathcal{S}$'s public key and stores this encryption at server $\mathcal{C}$. As the encryption scheme is additively homomorphic, $\mathcal{C}$ and $\mathcal{S}$ can run traditional SMC protocols to compute arbitrary functionalities, while the result is stored encrypted at $\mathcal{C}$. A client receives this result in the clear by running an interactive protocol with server $\mathcal{S}$ who helps in the decryption without learning anything at all. We note that similar approaches already appear in the literature (cf. Beye et al. [25] or Boneh et al. [26]).
2) **Secret Sharing Approach.** Every client secret shares its input and distributes it to the two servers, which can then compute any functionality in a secure two-party protocol. This is the secret sharing-based analogue of the previous single-key approach.

Although both constructions seem very promising, they actually solve a different problem scenario, as we have the explicit goal of allowing clients to encrypt their data under their *own* keys. The first approach even requires client interaction during the decryption which we want to circumvent. The second approach, besides the fact that it cannot deal with encryptions under different keys, has the additional drawback that both servers have the same (large) amount of storage and workload. Moreover, there are certain application scenarios (e.g., Private Smart Metering as we describe in Section VII) where the use of homomorphic encryption is very appealing.

C. Our Contribution

We provide a novel efficient technique (based on additively homomorphic encryption) that turns the above mentioned single-key approach into one that relies on no client interaction and works on encryptions under *multiple public keys* (in contrast to the above secret sharing approach). 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 [8], [11], [23], [24], [27]–[29], 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.

We make extensive use of the BCP cryptosystem by Bresson, Catalano and Pointcheval [30] 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 these inputs into encryptions under the

product of all involved public keys without changing the underlying plaintexts.

2) With these transformed ciphertexts (under the *same* 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.

Our approach is particularly suited to applications where clients are very resource-constrained and not always online such that having no interaction with these clients is of crucial importance. In this paper, we elaborate on two such applications from different domains. First, we deal with face recognition in social networks, where many users access their profiles via resource-constrained mobile devices that are not always online. In this setting, the tool of automated face recognition is used for image tagging services or in order to help law enforcement agencies to prosecute suspected persons. In order to show the feasibility of our approach, we implement an adaptation of Erkin et al.'s privacy-preserving face recognition protocol [29] to our construction and show a detailed performance analysis. Second, we consider the private aggregation of (energy) consumption in the smart metering scenario, where a distrusted (energy) supplier aggregates consumption readings from resource-constrained smart meters in a privacy-preserving manner [27], [28], [31]. As an additional contribution of our work and to highlight the versatility of our approach, we present three different variants of our construction that can be shown to be beneficial in such metering systems. Depending on the precise scenario, one of our variants allows for the intermediate (private) aggregation by other smart meters before the data arrives at the supplier. This is particularly interesting in the smart metering setting where sensor readings are relayed through other meters anyway due to the use of short-range communication networks. The other two variants deliberately sacrifice the non-interactive nature of our approach and allow the smart meters to have more control over their private data: interactively with *all* participating meters, the aggregated results can be disclosed either to the supplier only, or to all meters while the supplier learns nothing at all.

D. Outline

Some standard notation, the security model of semi-honest adversaries, and the BCP encryption scheme are summarized in Section II. We give our construction and proofs of correctness for the individual building blocks in Section III, while analyzing security in Section IV. Useful variants and opimizations of our protocols are given in Section V. We summarize our experimental results in Section VI and elaborate on application scenarios in Section VII. Finally, we conclude in Section VIII while focusing on possible future work.

II. Preliminaries

A. Notation, Security Model, and Access Control

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, $x \xleftarrow{U} X$ 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 [32]. This means that all involved parties follow the protocols, but try to gather information about 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 thus avoid cheating (which would cause negative publicity). 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.

Finally, we note that, similar to [22], it is possible to give each client more control over its private data by attaching access control policies to the encrypted inputs. These policies dictate what functions are allowed to be computed on the inputs and with what other inputs (from other clients) they are allowed to get combined. The enforcement of the policies would be done by the two servers $\mathcal{C}$ and $\mathcal{S}$. To simplify our exposition, we will not include such access control policies in the rest of this document.

B. 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: $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2)) = m_1 + m_2$, 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 as well as examples such as the Paillier encryption scheme [33], we refer to Armknecht et al. [34].

In this work, we use the additively homomorphic cryptosystem by Bresson, Catalano and Pointcheval (BCP) [30] which offers two independent decryption mechanisms (we note that this encryption scheme has been concurrently, but independently proposed by Camenisch and Shoup [35] and Damgård and Jurik [36]). 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 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.

The BCP Cryptosystem. The BCP encryption scheme [30] consists of the following algorithms:

$\mathsf{Setup}(\kappa)$, the master setup that generates the public parameters and the master secret: for 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', resp.) 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 the *public parameters* $\mathsf{PP} = (N, k, g)$ and the *master secret* $\mathsf{MK} = (p', q')$.

$\mathsf{KeyGen}(\mathsf{PP})$, the key generation algorithm that generates the users' public-secret key-pairs: pick a random $a \in \mathbb{Z}_{N^2}$ and compute $h = g^a \mod N^2$. The algorithm outputs the *public key* $\mathsf{pk} = h$ and the *secret key* $\mathsf{sk} = a$.

$\mathsf{Enc}_{(\mathsf{PP},\mathsf{pk})}(m)$, the encryption algorithm: given a user's public key pk and a plaintext $m \in \mathbb{Z}_N$, pick a random $r \in \mathbb{Z}_{N^2}$ and output the ciphertext (A, B) as

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

$\mathsf{Dec}_{(\mathsf{PP},\mathsf{sk})}(A, B)$, the "user" decryption of the cryptosystem which uses the user's secret key $\mathsf{sk} = a$ and only runs successfully on encryptions under the corresponding user's public key $\mathsf{pk} = g^a \mod N^2$: given a ciphertext (A, B) and secret key $\mathsf{sk} = a$, output the plaintext m as

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

$\mathsf{mDec}_{(\mathsf{PP},\mathsf{pk},\mathsf{MK})}(A, B)$, the "master" decryption which uses the master secret key MK and runs successfully on any properly created ciphertext, no matter under whose user's public key it was created: 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

$$a \mod N = \frac{h^{p'q'} - 1 \mod N^2}{N} \cdot k^{-1} \mod N, \tag{3}$$

where k^{-1} denotes the inverse of k modulo N. Then compute

$$r \mod N = \frac{A^{p'q'} - 1 \mod N^2}{N} \cdot k^{-1} \mod N. \tag{4}$$

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

$$m = \frac{(B/(g^\gamma))^{p'q'} - 1 \mod N^2}{N} \cdot \delta \mod N. \tag{5}$$

For proofs of correctness and semantic security (under the Decisional Diffie-Hellman assumption), we refer to [30]. 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 $\mathsf{Enc}_{(\mathsf{PP},\mathsf{pk})}(m)$.

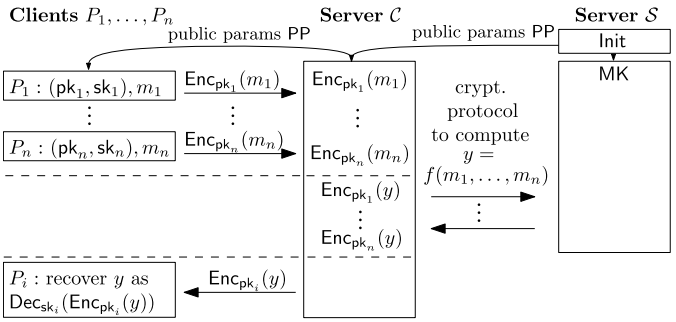

Fig. 1. Basic concept of our construction: Clients $P_1, \ldots, P_n$ upload encryptions of their private data $m_1, \ldots, m_n$ to server $\mathcal{C}$, which in turn engages in an SMC protocol with server $\mathcal{S}$ to compute $y = f(m_1, \ldots, m_n)$ in the encrypted domain. Once the protocol is done, each client P_i who is allowed to learn the result, is sent an encryption of y under its respective key.

III. 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 [37].

Our basic idea to realize this functionality can be summarized as follows (illustrated in Fig. 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 II and on the use of two non-colluding servers in the Introduction for further details on why this is a reasonable and worthwhile assumption.
2) Initially, $\mathcal{S}$ 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 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: $\mathsf{KeyProd}$, Add, 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

[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.

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 $f(m_1, \ldots, m_n)$.

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 $\mathcal{C}$.

Data Upload. In order to upload private data to the server $\mathcal{C}$, a client P_i (for some $i \in \{1, \ldots, n\}$) 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 $(\mathsf{pk}_i, \mathsf{sk}_i)$. After these keys are generated using algorithm KeyGen(PP) of the BCP cryptosystem, the client P_i can encrypt its private data m_i by computing $(A_i, B_i) \longleftarrow \mathsf{Enc}_{(\mathsf{PP},\mathsf{pk}_i)}(m_i)$ and upload it together with its public key pk_i to 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: KeyProd, Add, Mult and 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 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, 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. Multiplication gates can be securely evaluated by (an adapted version of) the well-known protocol of [4], 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 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.[2] The corresponding secret key required to successfully decrypt an encryption under 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 (according to the access control policies attached to each input) 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 Fig. 2.

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. Now, given two encryptions (A, B) and (A', B') under Prod.pk, server $\mathcal{C}$ computes the *sum* of their underlying plaintext messages by computing $(\overline{A}, \overline{B}) \longleftarrow (A \cdot A' \mod N^2, B \cdot B' \mod N^2)$.

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 [4]

[2]There is no particular reason why we use the product key here but it seems to be the most natural choice. Also, using this key allows us to do some efficiency tweaks in certain application scenarios (see Section V).

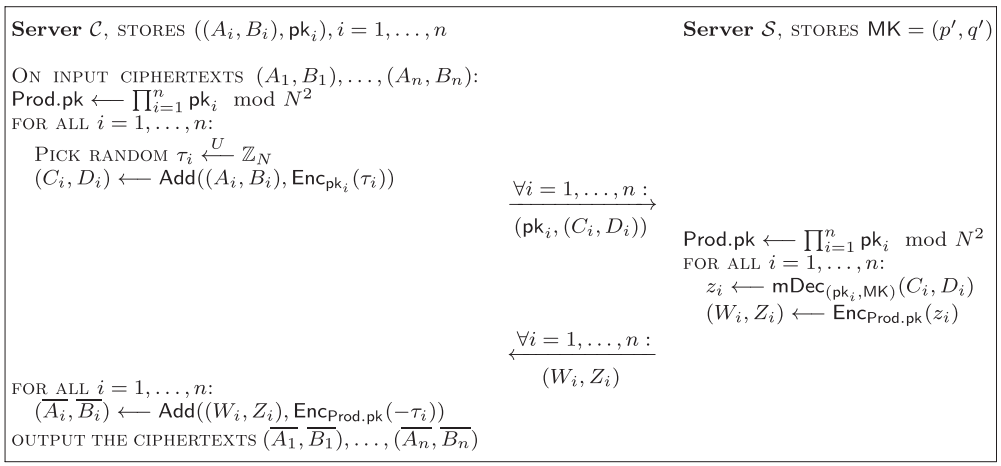

Fig. 2. KeyProd. Transforms encryptions under $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ into encryptions under $\prod_{i=1}^{n} \mathsf{pk}_i \mod N^2$ without changing the underlying plaintexts.

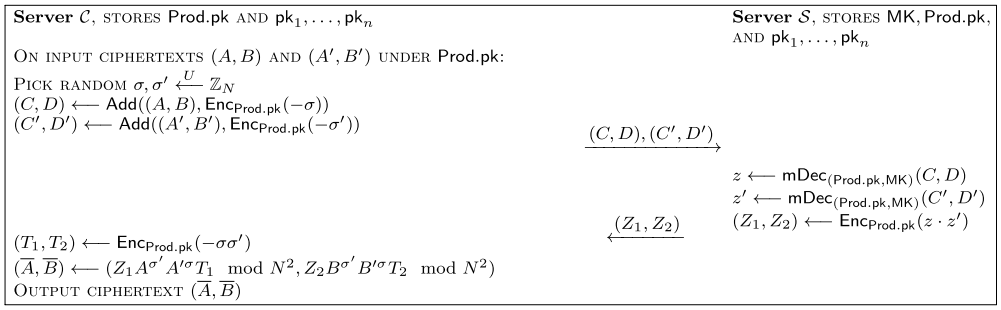

Fig. 3. Mult. Given two ciphertexts (A, B) and (A', B') under Prod.pk, it computes an encryption of the multiplication of the underlying plaintexts.

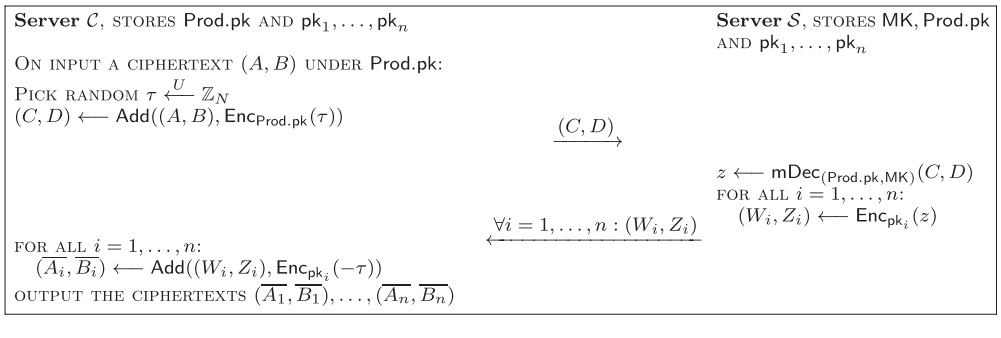

Fig. 4. TransDec. Transforms ciphertext (A, B) into n ciphertexts $(\overline{A_1}, \overline{B_1}), \ldots, (\overline{A_1}, \overline{B_1})$ of the same plaintext, encrypted under $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, respectively.

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 Fig. 3.

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 Fig. 4.

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 key pk_i that has been computed during the subprotocol TransDec, and then decrypting this ciphertext by using its corresponding private key sk_i.

IV. Correctness and Security

A. Correctness

We assume that all participants follow our protocol decriptions. Furthermore, we assume that the initial setup Init has been performed correctly, and that all clients (wishing to participate) sent their encrypted private data to the server $\mathcal{C}$ as described in the section on data upload. Under

these assumptions, we show that the remaining protocols KeyProd, Add, Mult and TransDec produce the desired outputs correctly.

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 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 [30] for details.

Mult: We show that the output ciphertext $(\overline{A}, \overline{B})$ of algorithm Mult is indeed an encryption (under Prod.pk) of the multiplication of the underlying plaintexts m, m' of the two input ciphertexts (A, B), (A', B') (encrypted under Prod.pk), respectively. Observe that the decryption of $(\overline{A}, \overline{B})$ under the secret key corresponding to 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. Fig. 3). 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.

TransDec: Recall that this protocol takes an encryption (A, B) under 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 keys pk_i, for $i = 1, \ldots, n$. 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, $i = 1, \ldots, n$.

B. Security

The following security analysis considers the semi-honest model only, as mentioned in Section II, 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 [32, 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 [32, 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, 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 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.

Adversarial Computing Power. We note that server $\mathcal{S}$ only sees blinded messages (after decryption). Since the used blinding values are chosen uniformly at random from the plaintext space $\mathbb{Z}_N$, the underlying messages are encrypted with a one-time-pad in $\mathbb{Z}_N$ with the random blinding values as secret keys. This type of encryption is perfect (i.e., information-theoretically) secure. Therefore, the computing power of server $\mathcal{S}$ (treated as an adversary) does not have to be bounded. At the same time, however, we require the computing power of $\mathcal{C}$ to be polynomially bounded as the BCP cryptosystem only offers semantic security in the presence of PPT adversaries.

V. Variants of the Construction

Recall that the main goal of our construction was to get rid of any interaction with the users. There are certain application

scenarios (cf. Section VII), 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 want to highlight the flexibility of our solution by giving three variants of our original proposal that allow leveraging it 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 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 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 (in the data upload), 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 KeyProd in this scenario: Recall, that KeyProd takes all the participating clients' ciphertexts (A_i, B_i) as input and transforms these to encryptions under the product 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 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 Fig. 5.

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

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.

Client Q_i HAS PUBLIC-PRIVATE KEY PAIR $(\mathsf{pk}_i, \mathsf{sk}_i)$

UPON RETRIEVAL OF CIPHERTEXT (A, B) (ENCRYPTED UNDER pk):
$B \longleftarrow A^{\mathsf{sk}_i} \cdot B \mod N^2$
OUTPUT CIPHERTEXT (A, B) (ENCRYPTED UNDER $\mathsf{pk} \cdot \mathsf{pk}_i \mod N^2$)

Fig. 5. Upon retrieval of a ciphertext (A, B) of m under pk, client Q_i produces an encryption of m under $\mathsf{pk} \cdot \mathsf{pk}_i \mod N^2$, $i = 1, \ldots, \ell$.

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 Fig. 6 *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. Hence, these modified versions of A can be used by $\mathcal{C}$ to "divide out" the public keys of each client seperately, ending up with an 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 Prod.pk, then the output m of ODApproval equals this message m'. If (A, B) was encrypted by using randomness r, then

$$(\overline{A}, \overline{B}) = (g^\tau \mod N^2, \mathsf{Prod.pk}^\tau (1 + m'N) \mod N^2), \quad (6)$$

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

$$m = \frac{(1 + m'N) - 1 \mod N^2}{N} = m'. \quad (7)$$

As for the security, we note that 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.

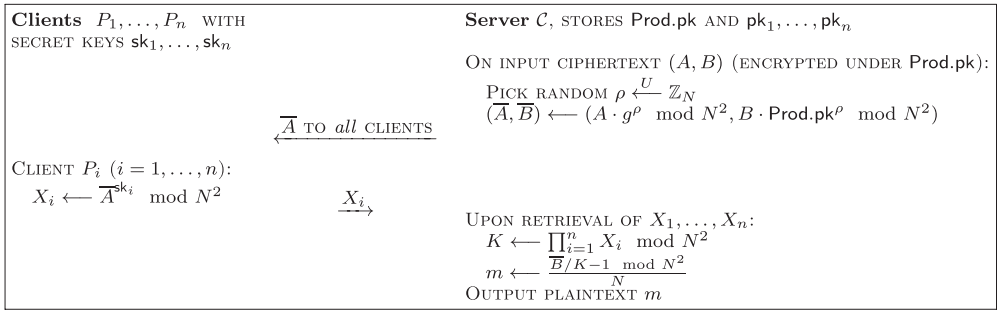

Fig. 6. ODApproval: Server $\mathcal{C}$ asks all involved clients for their approval to disclose the encrypted result to $\mathcal{C}$ while all clients stay oblivious to this result.

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 ODApproval (cf. Fig. 6). 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 Fig. 6) 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 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 $\mathcal{C}$.

VI. 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 Table I. 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.

TABLE I
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
KeyProd	$O(n\kappa^3)$ 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

The runtime of our code is heavily influenced by the size of N. We recommend a size of N of at least 1024 bits 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. Fig. 7(a) 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). Fig. 7(b) 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. This includes the encryption of inputs at the client side (sequentially). In practice, this step will be done in parallel by each client independently. Finally, we were interested in the runtime of Add and Mult. Fig. 7(c) 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. We stress again that we ignored network latency and

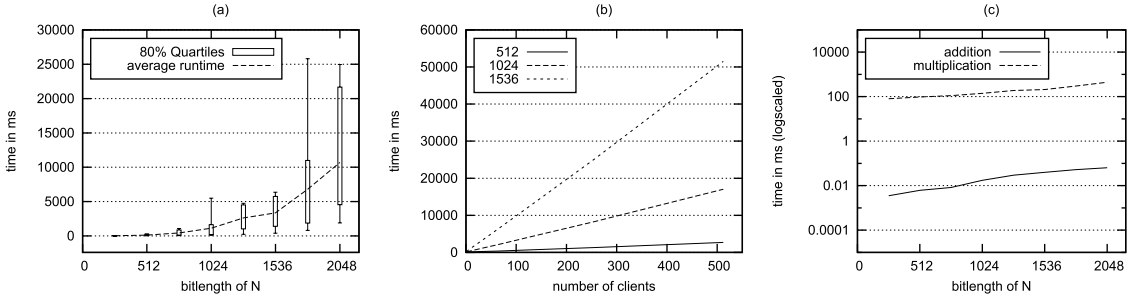

Fig. 7. Experimental results. Note that (b) includes the encryption of inputs at the client side.

that algorithm Mult runs interactively between servers $\mathcal{C}$ and $\mathcal{S}$. Therefore, when used in practice, the actual runtime of Mult will be even higher (depending on the latency).

Comparison With the Single-Key Approach. Assuming that the single-key construction (which cannot deal with encryptions under different keys, see Section I) uses Paillier encryption [33], the de facto standard in additively homomorphic encryption, and assuming no key transformations in our construction, we get approximately double the costs, both in terms of computation and communication. This is due to the fact that ciphertexts in the BCP scheme have double the size of Paillier encryptions, while encryptions and decryptions (taking pre-computations into account) require one additional exponentiation modulo N^2, and additions in the encrypted domain require one additional multiplication modulo N^2. This shows that our approach achieves computation on encrypted data across different keys while requiring only little additional costs.

VII. Applications

We present two applications of our general-purpose construction. First, we adapt the privacy-preserving face recognition protocol by Erkin et al. [29] to our technique and provide a performance analysis to show the feasibility of our approach. Second, we apply our variants from Section V to private consumption aggregation in the smart metering setting. We stress that we are not providing entire solutions to these two application scenarios, but solely want to demonstrate the potential of our construction in such scenarios, which both would benefit from the non-inter-active nature of our protocols.

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 or in order to help law enforcement agencies to prosecute suspected persons (to name just a few). This is usually being done without the explicit consent of users, raising important privacy concerns, as convincingly demonstrated in [29]. Therefore, Erkin et al. [29] proposed the first privacy-preserving 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. In order for such a system to be completely non-interactive, it is required that users have their own public keys under which the results of the face recognition protocol will be encrypted.

Similarly to Erkin et al. [29], 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.[3] To achieve this goal, we adapt the privacy-preserving protocol by Erkin et al. [29] to our framework.

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. However, all such tricks can be applied to our protocols in exactly the same way as in [29]. For instance, switching to the DGK cryptosystem is done when determining whether a small value d that server $\mathcal{S}$ knows in the clear is less or greater-than-or-equal to a small value r that $\mathcal{C}$ holds in the clear. This comparison is the computational bottleneck of the whole face recognition protocol. The DGK cryptosystem provides a way to do this comparison very efficiently as long as the values d and r are small. Since $\mathcal{C}$ and $\mathcal{S}$ know the values in the clear, Erkin et al. [29] propose that $\mathcal{S}$ runs the DKG key generation and sends the public key to $\mathcal{C}$. Then, $\mathcal{C}$ and $\mathcal{S}$ perform the comparison DGK-encrypted while at the end of the protocol, $\mathcal{S}$ learns a DGK-encrypted blinded bit b indicating whether $d < r$ or $d \geq r$, while $\mathcal{C}$ knows the blinding value (which is a bit itself). In [29], $\mathcal{S}$ decrypts this blinded bit b and then encrypts it again with the Paillier encryption scheme [33], sends the result to $\mathcal{C}$ who in turn removes the blinding value. We note that $\mathcal{C}$'s knowledge of the Paillier encrypted bit b is enough to perform the remaining steps of

[3]Note that the correct recognition is only assured with a certain probability, since the tool of face recognition is error-prone.

TABLE II
COMPLEXITY OF PERFORMING PRIVACY-PRESERVING FACE RECOGNITION. 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 HAD HIGHER PERFORMANCE THAN THE LOCAL ONE

Database size M	Runtime in sec.	Traffic $\mathcal{C}$ to $\mathcal{S}$ in MB	Traffic $\mathcal{S}$ to $\mathcal{C}$ in MB	Runtime w/ remote $\mathcal{S}$ 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

the comparison used in the face recognition protocol (see [29] for details). Adapting this approach to our system would boil down to the replacement of the Paillier encryption scheme with the BCP scheme in the final step where $\mathcal{S}$ re-encrypts the blinded bit b. We did not implement these tricks to show the performance of our general-purpose construction in its plain as-is state without tweaking it to specific application scenarios. We implemented the complete protocol in python while basing it on our implementation of our general-purpose construction described in Section VI. All tests were performed in the same setting as in Section VI with the security parameter $\kappa = 1024$.

To avoid confusion, we used the same parameters for our implementation as [29]. In particular, we also used the "ORL Database of Faces" from AT&T Laboratories Cambridge [38] 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 $L = 92 \cdot 112 = 10304$. This setup ensures the same reliability results as in [29], which obtains a correct classification rate of $\approx 96\%$ in the identification setting using the standard Eigenfaces recognition algorithm by Turk and Pentland [39]. We stress that we achieve such a high classification rate as the ORL database contains images from a controlled environment and it will be much lower for real social network profile pictures. It is demonstrated in [29] that $K > 12$ eigenfaces in the enrolment phase do not significantly increase the correct classification rate in case of the ORL database, which is why we used their suggestion of $K = 12$ (for $M = 10$, we used $K = 5$ instead).

Table II depicts the results of our performance analysis of the complete 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 data traffic (in megabytes) sent from $\mathcal{C}$ to $\mathcal{S}$, while the fourth 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.

In summary, for a database of size $M = 200$, a full protocol run requires $\approx$ 16 minutes (both when running $\mathcal{S}$ locally or remotely) for checking whether a given encrypted image is contained in the database or not. Again, we stress that we can apply the same performance tweaks to our protocol as done in [29], which would yield a significant efficiency boost. More techniques to further enhance the performance in face recognition can also be found in [40].

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. Depending on the scenario, it should also be possible for the supplier to send out customized recommendations on the basis of (encrypted) data from other meters in a way such that only the receiving smart meter is able to see the recommendation.

Here, we focus on private consumption aggregation only (and the possibility to send out recommendations on the basis of these aggregations) while dealing with a semi-honest supplier. In this setting, current solutions either only consider eavesdropping outsiders (and not the supplier) [28], allow the supplier to see certain results (e.g., intermediate aggregations [41]), require interaction and high computational overhead at the smart meters [27], [31], rely on trusted components alongside each smart meter [42], [43], or accept a certain decrease in the utility of the aggregated results by using differentially private mechanisms [44] (their system is also very restrictive in what kind of aggregations can be performed). We stress that we are not concerned with neither fraud detection nor the non-repudiation, integrity, or verifiability of aggregated results (see [31] for a treatment of these aspects).

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

Our construction of Section III offers a protocol with low computational costs at and minimal communication with the smart meters. As recently noted in [45], achieving a solution with minimal communication overhead at the smart meters is one of the great research challenges. We hope that our approach provides another step forward to cope with this challenge. The smart meters would act as the clients in our protocol description of Section III, 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. Compared to the above mentioned existing works that can hide the aggregated result (and intermediate results) from the supplier, our approach offers the unique feature of requiring no interaction with the meters during computations while allowing all kind of different aggregations (and not just additions). Exceptions are the approaches [42] and [43] that require trusted components at the smart meters, and the approach in [44] using differentially private mechanisms which comes at the cost of decreased utility and restricted aggregation methods.

Furthermore, concerning private consumption aggregation, we can extend the distributed incremental data aggregation approach of [28] in order to protect the consumers' privacy from the supplier $\mathcal{C}$ as well. In [28], 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 [28], 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 [28]) 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) use the variant "Disclosure by Clients' Approval" meaning that $\mathcal{C}$ can only decrypt by all consumers' approval;
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.

VIII. Conclusion

Assuming the existence of two semi-honest and non-colluding but otherwise untrusted servers, we presented a new technique for the efficient and secure outsourcing of general-purpose SMC protocols between n users to these two servers. Our technique is based on the BCP encryption scheme, requires no user interaction at all, and allows the evaluation of arbitrary computable functions on inputs encrypted under different public keys. We show our protocol to be secure in the semi-honest model and highlight its practicability by giving experimental results. The shown possibility of employing our technique in two application scenarios, namely on privacy-preserving face recognition and private consumption aggregation in smart metering, underlines its feasibility and versatility.

We consider the extension to the malicious adversarial model as future work and we plan to experiment with certain functionalities found in other application scenarios, e.g., in genomic processing [46]. To this end, it seems useful to also integrate our work into tools for automating SMC [47].

References

[1] B. Worley, (2010, Oct. 19). *The Facebook Privacy Scandal* [Online]. Available: http://abcnews.go.com/GMA/Consumer/facebook-privacy-scandal-facebooks-w%atergate/story?id=11912201

[2] M. Barbaro and T. Zeller, (2006, Aug. 9). *A Face is Exposed for AOL Searcher no. 4417749* [Online]. Available: http://www.nytimes.com/2006/08/09/technology/09aol.html

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM CCS*. 2009, pp. 199–212.

[4] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Proc. Int. Conf. Theory Appl. Cryptograph. Tech., Adv. Cryptol.*, 2001, pp. 280–299.

[5] S. Jarecki and V. Shmatikov, "Efficient two-party secure computation on committed inputs," in *Proc. 26th Annu. Int. Conf. Adv. Cryptol.*, 2007, pp. 97–114.

[6] A. C.-C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, 1982, pp. 160–164.

[7] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Proc. 26th Annu. Int. Conf. Theory Appl. Cryptograph. Tech.*, 2007, pp. 52–78.

[8] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Proc. ASIACRYPT*, 2009, pp. 250–267.

[9] I. Damgård, Y. Ishai, and M. Krøigaard, "Perfectly secure multiparty computation and the computational overhead of cryptography," in *Proc. EUROCRYPT*, 2010, pp. 445–465.

[10] E. Boyle, S. Goldwasser, A. Jain, and Y. T. Kalai, "Multiparty computation secure against continual memory leakage," in *Proc. STOC*, 2012, pp. 1235–1254.

[11] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, *et al.*, "Secure multiparty computation goes live," in *Proc. 13th Int. Conf. FC*, 2009, pp. 325–343.

[12] I. Damgård and Y. Ishai, "Constant-round multiparty computation using a black-box pseudorandom generator," in *Proc. CRYPTO*, 2005, pp. 378–394.

[13] U. Feige, J. Kilian, and M. Naor, "A minimal model for secure computation," in *Proc. 26th Annu. ACM Symp. Theory Comput.*, 1994, pp. 554–563.

[14] Y. Ishai and E. Kushilevitz, "Private simultaneous messages protocols with applications," in *Proc. 5th ISTCS*, 1997, pp. 174–184.

[15] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," in *Proc. IACR*, 2011, pp. 1–41.

[16] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proc. ACM Conf. Electron. Commerce*, 1999, pp. 129–139.
[17] S. G. Choi, A. Elbaz, A. Juels, T. Malkin, and M. Yung, "Two-party computing with encrypted data," in *Proc. ASIACRYPT*, 2007, pp. 298–314.
[18] S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the web: Computing without simultaneous interaction," in *Proc. CRYPTO*, 2011, pp. 132–150.
[19] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.
[20] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Adv. CRYPTO*, 2011, pp. 505–524.
[21] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. 44th Symp. Theory Comput.*, 2013, pp. 1219–1234.
[22] M. Van Dijk and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing," in *Proc. HotSec Cryptol. ePrint Archive*, 2010, pp. 1–8.
[23] O. Catrina and F. Kerschbaum, "Fostering the uptake of secure multiparty computation in E-commerce," in *Proc. 3rd Int. Conf. ARES*, 2008, pp. 693–700.
[24] Z. Erkin, T. Veugen, T. Toft, and R. Lagendijk, "Generating private recommendations efficiently using homomorphic encryption and data packing," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 3, pp. 1053–1066, Jun. 2012.
[25] M. Beye, Z. Erkin, and R. L. Lagendijk, "Efficient privacy preserving K-means clustering in a three-party setting," in *Proc. IEEE Int. WIFS*, Dec. 2011, pp. 1–6.
[26] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Proc. ACNS*, 2013, pp. 102–118.
[27] F. D. Garcia and B. Jacobs, "Privacy-friendly energy-metering via homomorphic encryption," in *Proc. STM*, 2010, pp. 226–238.
[28] F. Li, B. Luo, and P. Liu, "Secure and privacy-preserving information aggregation for smart grids," *IJSN*, vol. 6, no. 1, pp. 28–39, 2011.
[29] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Proc. PET*, 2009, pp. 235–253.
[30] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Proc. ASIACRYPT*, 2003, pp. 37–54.
[31] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *Proc. PETS*, 2011, pp. 175–191.
[32] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge U.K.: Cambridge Univ. Press, 2004.
[33] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. EUROCRYPT*, 1999, pp. 223–238.
[34] F. Armknecht, S. Katzenbeisser, and A. Peter, "Group homomorphic encryption: Characterizations, impossibility results, and applications," *Des. Codes Cryptograph.*, vol. 67, no. 2, pp. 209–232, 2013.
[35] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *Proc. CRYPTO*, 2003, pp. 126–144.
[36] I. Damgård and M. Jurik, "A length-flexible threshold cryptosystem with applications," in *Proc. ACISP*, 2003, pp. 350–364.
[37] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design," in *Proc. IACR Cryptol. ePrint Archive*, 2010, pp. 1–20.
[38] (1992). *The Database of Faces, (Formerly 'the orl Database of Faces')* [Online]. Available: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[39] M. Turk and A. Pentland, "Eigenfaces for recognition," *J. Cognitive Neurosci.*, vol. 3, no. 1, pp. 71–86, Jan. 1991.
[40] J. Bringer, H. Chabanne, and A. Patey, "Privacy-preserving biometric identification using secure multiparty computation: An overview and recent trends," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 42–52, Mar. 2013.
[41] Z. Erkin and G. Tsudik, "Private computation of spatial and temporal power consumption with smart meters," in *Proc. ACNS*, 2012, pp. 561–577.
[42] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin, "Private memoirs of a smart meter," in *Proc. 2nd ACM BuildSys*, 2010, pp. 61–66.
[43] M. Jawurek, M. Johns, and F. Kerschbaum, "Plug-in privacy for smart metering billing," in *Proc. PETS*, 2011, pp. 192–210.
[44] G. Ács and C. Castelluccia, "I have a dream! (differentially private smart metering)," in *Proc. IH*, 2011, pp. 118–132.
[45] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Pérez-González, "Privacy-preserving data aggregation in smart metering systems: An overview," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 75–86, Mar. 2013.
[46] P. Baldi, R. Baronio, E. D. Cristofaro, P. Gasti, and G. Tsudik, "Countering gattaca: Efficient and secure testing of fully-sequenced human genomes," in *Proc. 18th ACM CCS*, 2011, pp. 691–702.
[47] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith, "Secure two-party computations in ANSI C," in *Proc. CCS*, 2012, pp. 772–783.

Andreas Peter received the Ph.D. degree in computer science from the Technical University of Darmstadt, a German diploma in mathematics from the Carl-von-Ossietzky University of Oldenburg, and the M.A.St. degree in mathematics from the University of Cambridge, U.K. Since March 2013, he is has been a post-doctoral with the Services, Cybersecurity and Safety Group, University of Twente. His research interests lie in the area of applied cryptography, focusing on applications of secure multiparty computation, searchable encryption, and homomorphic encryption.

Erik Tews received the Ph.D. degree from the Technical University of Darmstadt about the security of cordless phones (DECT Phones) in 2011. This included parts from different research areas like radio protocols, reverse engineering, FPGA and micro controller programming, as well as cryptanalysis, designing attacks and countermeasures against cryptographic primitives, and the analysis of the DECT radio protocol. He received the German diploma and bachelor's degrees from the Technical University of Darmstadt. His final thesis was about the security of WEP protected wireless LAN networks, including the design and implementation of an advanced attack against the RC4 stream cipher.

Stefan Katzenbeisser (S'98–A'01–M'07–SM'12) received the Ph.D. degree from the Vienna University of Technology, Austria. After working as a Research Scientist with the Technical University in Munich, Germany, he joined Philips Research as a Senior Scientist in 2006. Since April 2008, he has been a Professor with the Technical University of Darmstadt, heading the Security Engineering Group. His current research interests include digital rights management, data privacy, software security, and cryptographic protocol design. He has authored more than 100 scientific publications and served on the program committees of several workshops and conferences devoted to information security. He was a Program Chair of the Information Hiding in 2005, General Chair of the 5th International Conference on Trust and Trustworthy Computing (2012), and Workshop Chair of the ACM Conference on Computer and Communications Security (2013). Between 2009 and 2011, he served on the Information Forensics and Security Technical Committee of the IEEE Signal Processing Society.