Outsourced Pattern Matching

Sebastian Faust[1,⋆], Carmit Hazay[2], and Daniele Venturi[3,⋆⋆]

[1] Security and Cryptography Laboratory, EPFL, Switzerland
[2] Faculty of Engineering, Bar-Ilan Univeristy, Israel
[3] Department of Computer Science, Aarhus University, Denmark

Abstract. In secure delegatable computation, computationally weak devices (or clients) wish to outsource their computation and data to an *untrusted* server in the cloud. While most earlier work considers the general question of how to securely outsource *any* computation to the cloud server, we focus on *concrete* and *important* functionalities and give the first protocol for the *pattern matching* problem in the cloud. Loosely speaking, this problem considers a text T that is outsourced to the cloud S by a client C_T. In a query phase, clients $C_1, \ldots, C_l$ run an efficient protocol with the server S and the client C_T in order to learn the positions at which a pattern of length m matches the text (and nothing beyond that). This is called the *outsourced pattern matching* problem and is highly motivated in the context of delegatable computing since it offers storage alternatives for massive databases that contain confidential data (e.g., health related data about patient history). Our constructions offer *simulation-based security* in the presence of semi-honest and malicious adversaries (in the random oracle model) and limit the communication in the query phase to $O(m)$ bits plus the number of occurrences — which is optimal. In contrast to generic solutions for delegatable computation, our schemes do not rely on fully homomorphic encryption but instead uses novel ideas for solving pattern matching, based on efficiently solvable instances of the subset sum problem.

1 Introduction

The problem of securely outsourcing computation to an *untrusted* server gained momentum with the recent penetration of *cloud computing* services. In cloud computing, clients can lease computing services on demand rather than maintaining their own infrastructure. While such an approach naturally has numerous advantages in cost and functionality, the outsourcing mechanism crucially needs to enforce privacy of the outsourced data and integrity of the computation. Cryptographic solutions for these challenges have been put forward with the concept of *secure delagatable computation* [1,6,11,2,8].

⋆ Supported in part by the BEAT project 7th Framework Research Programme of the European Union, grant agreement number: 284989.

⋆⋆ Supported from the Danish National Research Foundation, the National Science Foundation of China (under the grant 61061130540), the Danish Council for Independent Research (under the DFF Starting Grant 10-081612) and also from the CFEM research center within which part of this work was performed.

In secure delegatable computation, computationally weak devices (or clients) wish to outsource their computation and data to an *untrusted* server. The ultimate goal in this setting is to design efficient protocols that minimize the computational overhead of the clients and instead rely on the extended resources of the server. Of course, the amount of work invested by the client in order to verify the correctness of the computation shall be *substantially* smaller than running the computation by itself. Indeed, if this was not the case then the client could carry out the computation itself. Another ambitious goal of delegatable computation is to design protocols that minimize the *communication* between the cloud and the client.

Most recent works in the area of delegatable computation propose solutions to securely outsource *any functionality* to an untrusted server [1,6,11,2]. Such generic solutions often suffer from rather poor efficiency and high communication overhead due to the use of fully homomorphic encryption [12]. An exception is the randomized encoding technique used by [1] which instead relies on garbled circuits. Furthermore, these solution concepts typically examine a restricted scenario where a *single client* outsources its computation to an *external untrusted server.* Only few recent works study the setting with *multiple clients* that mutually distrust each other and wish to securely outsource a joint computation on their inputs with reduced costs, e.g., [15,17]. Of course, also in this more complex setting recent constructions build up on fully homomorphic encryption.

To move towards more practical schemes, we may focus on particularly efficient constructions for specific important functionalities. This approach has the potential to avoid the use of fully homomorphic encryption by exploiting the structure of the particular problem we intend to solve. Some recent works have considered this question [3,22,20]. While these schemes are more efficient than the generic constructions mentioned above, they typically only achieve very limited privacy or do not support multiple distrusting clients. In this paper, we follow this line of work and provide the first protocols for *pattern matching* in the cloud. In contrast to most earlier works, our constructions achieve a high-level of security, while avoiding the use of FHE and minimizing the amount of communication between the parties. We emphasize that even with the power of fully homomorphic encryption it is not clear how to get down to communication complexity that is linear in the number of matches in two rounds.[1]

Pattern Matching in the Cloud. The problem of pattern matching considers a text T of length n and a pattern of length m with the goal to find all the locations where the pattern matches the text. In a secure pattern matching protocol, one party holds the text whereas the other party holds the pattern and attempts to learn all the locations of the pattern in the text (and only that), while the party holding the text learns nothing about the pattern. Unfortunately, such protocols are not directly applicable in the cloud setting, mostly because the

[1] A one-round solution based on FHE would need a circuit that tolerates the maximal number of matches — which in the worst case is proportional to the length of the text.

communication overhead per search query grows linearly with the text length. Moreover, the text holder delegates its work to an external untrusted server and cannot control the content of the server's responses.

In the outsourced setting we consider a set of clients $C_T, (C_1, \ldots, C_l)$ that interact with a server S in the following way. (**1**) In a *setup phase* client C_T uploads a preprocessed text to an external server S. This phase is run only once and may be costly in terms of computation and communication. (**2**) In a *query phase* clients $C_1, \ldots, C_l$ query the text by searching patterns and learn the matched text locations. The main two goals of our approach are as follows:

1. *Simulation-based security:* We model outsourced pattern matching by a strong simulation-based security definition (cf. Section 2). Namely, we define a new reactive outsourced functionality $\mathcal{F}_{\mathrm{OPM}}$ that ensures the secrecy and integrity of the outsourced text and patterns. For instance, a semi-honest server does not gain any information about the text and patterns, except of what it can infer from the answers to the search queries. If the server is maliciously corrupted the functionality implies the correctness of the queries' replies as well. As in the standard secure computation setting, simulation-based modeling is simpler and stronger than game-based definitions.
2. *Sublinear communication complexity during query phase:* We consider an *amortized* model, where the communication and computational costs of the clients are reduced with the number of queries. More concretely, while in the setup phase communication and computation is linear in the length of the text, we want that during the query phase the overall communication and the work put by the clients is *linear in the number of matches* (which is optimal). Of course, we also require the server running in polynomial-time. Clearly, such strong efficiency requirement comes at a price as it allows the server to learn the number of matches. We model this additional information by giving the server *some leakage* for each pattern query which will be described in detail below.

1.1 Our Contribution

To simplify notation we will always only talk about a single client C that interacts with C_T and S in the query phase.

Modeling Outsourced Pattern Matching. We give a specification of an ideal execution with a trusted party by defining a reactive outsourced pattern matching functionality $\mathcal{F}_{\mathrm{OPM}}$. This functionality works in two phases: In the preprocessing phase client C_T uploads its preprocessed text $\widetilde{T}$ to the server. Next, in an iterative query phase, upon receiving a search query p the functionality asks for the approvals of client C_T (as it may also refuse for this query in the real execution), and the server (as in case of being corrupted it may abort the execution). To model the additional leakage that is required to minimize communication we ask the functionality to forward to the server the matched positions in the text upon receiving an approval from C_T. Our functionality returns all matched positions but can be modified so that only the first few matched positions are returned.[2]

Difficulties with Simulating $\mathcal{F}_{\mathrm{OPM}}$. The main challenge in designing a simulator for this functionality is in case when the server is corrupted. In this case the simulator must *commit* to some text in a way that later allows him (when taking the role of the server, given some trapdoor) to reply to pattern queries in a consistent way. More precisely, when the simulator commits to a preprocessed text, the leakage that the corrupted server obtains (namely, the positions where the pattern matches the text) has to be consistent with the information that it later sees during the query phases. This implies that the simulator must have flexibility when it later matches the committed text to the trapdoors. This difficulty does not arise in the classic two-party setting since there the simulator always plays against a party that contributes an input to the computation which it can first extract, whereas here the server is just a tool to run a computation. Due to this inherent difficulty the text must be *encoded* in a way, that given a search query p and a list of text positions $(i_1, \ldots, i_t)$, one can produce a trapdoor for p in such a way that the "search" in the preprocessed text, using this trapdoor, yields $(i_1, \ldots, i_t)$. We note that alternative solutions that permute the text to prevent the server from learning the matched positions, necessarily require that the server does not collude with the clients. In contrast, our solutions allow such strong collusion between the clients and the server.

Solutions Based on Searchable/Non-Committing Encryption. To better motivate our solution, let us consider a toy example first. Assume we encrypt each substring of length m in T using searchable encryption [4], which allows running a search over an encrypted text by producing a trapdoor for the searched word (or a pattern p). Given the trapdoor, the server can check each ciphertext and return the text positions in which the verification succeeds. The first problem that arises with this approach is that searchable encryption does not ensure the privacy of the searched patterns. While this issue may be addressed by tweaking existing constructions of searchable encryption, a more severe problem is that the simulator must commit in advance to (searchable) encryptions of a text that later allow to "find" p at positions that are consistent with the leakage. In other words: all the plaintexts in the specified positions must be associated with the keyword p ahead of time. Of course, as the simulator does not know the actual text T it cannot produce such a consistent preprocessed text. An alternative solution may be given by combining searchable encryption with techniques from non-committing encryptions [5]. Note that it is unclear how to combine these two tools even in the random oracle model.

[2] This definition is more applicable for search engines where the first few results are typically more relevant, whereas the former variant is more applicable for a DNA search where it is important to find all matched positions. For simplicity we only consider the first variant, our solutions support both variants.

Semi-Honest Outsourced Pattern Matching from Subset Sum. Our first construction for outsourced pattern matching is secure against semi-honest adversaries. In this construction client C_T generates a vector of random values, conditioned on that the sum of elements in all positions that match the pattern equals some specified value that will be explained below. Namely, C_T builds an instance $\widetilde{T}$ for the *subset sum problem*, where given a trapdoor R the goal is to find whether there exists a subset in $\widetilde{T}$ that sums to R. More formally, the subset sum problem is parameterized by two integers ℓ and M. An instance of the problem is generated by picking random vectors $\widetilde{T} \leftarrow \mathbb{Z}_M^\ell$, $\mathbf{s} \leftarrow \{0,1\}^\ell$ and outputting $(\widetilde{T}, R = \widetilde{T} \cdot \mathbf{s} \bmod M)$. The problem is to find $\mathbf{s}$ given $\widetilde{T}$ and a trapdoor R. Looking ahead, we will have such a trapdoor R_p for each pattern p of length m, such that if p matches T then with overwhelming probability there will be a unique solution to the subset sum instance $(\widetilde{T}, R_p)$. This unique solution is placed at exactly the positions where the pattern appears in the text. The client C that wishes to search for a pattern p obtains this trapdoor from C_T and will hand it to the server. Consequently, we are interested in easy instances of the subset sum problem since we require the server to solve it for each query. This is in contrast to prior cryptographic constructions, e.g., [18] that design cryptographic schemes based on the hardness of this problem. We therefore consider low-density instances which can be solved in polynomial time by a reduction to a short vector in a lattice [16,10,7].

We further note that the security of the scheme relies heavily on the unpredictability of the trapdoor. Namely, in order to ensure that the server cannot guess the trapdoor for some pattern p (and thus solve the subset problem and find the matched locations), we require that the trapdoor is unpredictable. We therefore employ a pseudorandom function (PRF) F on the pattern and fix this value as the trapdoor, where the key k for the PRF is picked by C_T and the two clients C_T and C communicate via a secure two-party protocol to compute the evaluation of the PRF.

Efficiency Considerations. The scheme described above does not yet satisfy the desired properties outlined in the previous paragraphs and has a very limited usage in practice. Recall that the server is asked to solve subset sum instances of the form $(\widetilde{T}, R_p)$, where $\widetilde{T}$ is a vector of length $\ell = n - m + 1$ with elements from $\mathbb{Z}_M$ for some integer M. In order to ensure correctness we must guarantee that given a subset sum instance, each trapdoor has a unique solution with high probability. In other words, the collision probability, which equals $2^\ell/M$ (stated also in [13]), should be negligible. Fixing $M = 2^{\kappa+n}$ for a security parameter κ, ensures this for a large enough κ, say whenever $\kappa \geq 80$. On the other hand, we need the subset sum problem to be solvable in polynomial time. A simple calculation (see Eq. (1)), yields in this case a value of $\ell \approx \sqrt{\kappa}$. This poses an inherent limitation on the length of the text to be preprocessed. For instance, even using a high value of $\kappa \approx 10^4$ (yielding approximately subset sum elements of size 10 KByte) limits the length of the text to only 100 bits. This scheme also requires quadratic communication complexity in the text length during the setup phase since client C_T sends $O(n^2 + \kappa n)$ bits.

An Improved Solution Using Packaging. To overcome this limitation, we employ an important extension of our construction based on *packaging*. First, the text is partitioned into smaller pieces of length $2m$ which are handled separately by the protocol, where m is some practical upper bound on the pattern length. Moreover, every two consecutive blocks are overlapping in m positions, so that we do not miss any match in the original text. Even though this approach introduces some overhead, yielding a text T' of overall length $2n$, note that now Eq. (1) yields $\ell = 2m - m + 1 = m + 1 < \sqrt{\kappa}$, which is an upper bound on the length of the pattern (and not on the length of the text as before). Namely, we remove the limitation on the text length and consider much shorter blocks lengths for the subset sum algorithm. As a result, the communication complexity in the setup phase is $O(mn + \kappa n)$, whereas the communication complexity in the query phase is $O(\kappa m)$. For short queries (which is typically the case), these measures meet the appealing properties we are sought after.

This comes at a price though since we now need to avoid using in each block the same trapdoor for some pattern p, as repetitions allow the server to extract potential valid trapdoors (that have not been queried yet) and figure out information about the text. We solve this problem by requiring from the function outputting the trapdoors to have some form of "programmability" (which allows to simulate the answers to all queries consistently). Specifically, we implement this function using the random oracle methodology on top of the PRF, so that a trapdoor now is computed by $\mathcal{H}(\mathsf{F}(k,p)\|b)$, for b being the block number. Now, the simulator can program the oracle to match with the positions where the pattern appears in each block. Note that using just the random oracle (without the PRF) is not sufficient as well, since an adversary that controls the server and has access to the random oracle can apply it on p as well.

Malicious Outsourced Pattern Matching. We extend our construction to the malicious setting as well, tolerating malicious attacks. Our proof ensures that the server returns the correct answers by employing Merkle commitments and zero-knowledge (ZK) sets. Informally speaking, Merkle commitments are succinct commitment schemes for which the commitment size is independent of the length of the committed value (or set). This tool is very useful in ensuring correctness, since now, upon committing to $\widetilde{T}$, the server decommits the solution to the subset sum trapdoor and client C can simply verify that the decommitted values correspond to the trapdoor. Nevertheless, this solution does not cover the case of a mismatch. Therefore, a corrupted server can always return a "no-match" massage. In order to resolve this technicality we borrow techniques from ZK sets arguments [19], used for proving whether an element is in a specified set without disclosing any further information. Next, proving security against a corrupted C is a straightforward extension of the semi-honest proof using the modifications we made above and the fact that the protocol for implementing the oblivious PRF evaluation is secure against malicious adversaries as well.

The case of a corrupted C_T is more challenging since we first need to extract the text T, but also verify C_T's computations with respect to the random oracle when it produces $\widetilde{T}$. The only proof technique that we are aware of for proving correctness when using a random oracle is cut-and-choose (e.g., as done in [14]), which inflates the communication complexity by an additional statistical parameter. Instead, we do not require that the server can verify immediately the correctness of the outsourced text, but only ensure that if C_T cheats with respect to some query p, then it will be caught during the query phase whenever p is queried. The crux of our protocol is that the simulator does not need to verify all computations at once, but only the computations with respect to the asked queries. This enables us to avoid the costly cut-and-choose technique since verification is done using a novel technique of *derandomizing* C_T's computations. We notice that this requires us to slightly adjust the description of our idealized functionality. For space reasons, we defer the details to the full version [9] and focus on the semi-honest case here.

We remark that all the solutions described above can be combined together into a single protocol which is secure even in the case of a collusion between S and client C. When a collusion between S and client C_T occurs we cannot guarantee either privacy or correctness since the simulator cannot extract the text, as the preprocessed protocol is "run" between the two corrupted parties. We stress that collusion does not imply that security collapses into the standard two-party setting.

2 Modeling Outsourced Pattern Matching

The outsourced pattern matching consists of two phases. In the *setup phase* a client C_T uploads a (preprocessed) text $\widetilde{T}$ to an external server S. This phase is run only once. In the *query phase* client C queries the text by searching patterns and learn the matched text locations. We formalize security using the ideal/real paradigm. Denote by T_j the substring of length m that starts at text location j. The pattern matching ideal functionality in the outsourced setting is depicted in Fig. 1. We write $|T|$ for the bit length of T and assume that client C asks a number of queries p_i ($i \in [\lambda]$, $\lambda \in \mathbb{N}$).

The Definition. Formally, denote by $\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}},\mathsf{Sim}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))$ the output of an ideal adversary Sim, server S and clients C_T, C in the above ideal execution of $\mathcal{F}_{\mathrm{OPM}}$ upon inputs $(-, (T, (p_1, \ldots, p_\lambda)))$ and auxiliary input z given to Sim.

We implement functionality $\mathcal{F}_{\mathrm{OPM}}$ via a protocol $\pi = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$ consisting of three two-party protocols, specified as follows. Protocol π_{Pre} is run in the preprocessing phase by C_T to preprocess text T and forward the outcome $\widetilde{T}$ to S. During the query phase, protocol π_{Query} is run between C_T and C (holding a pattern p); this protocol outputs a trapdoor R_p that depends on p and will enable the server to search the preprocessed text. Lastly, protocol π_{Opm} is run by S upon input the preprocessed text and trapdoor R_p (forwarded by C); this protocol returns C the matched text positions (if any). We denote by $\mathbf{REAL}_{\pi,\mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))$ the output of adversary Adv, server S and clients C_T, C in a real execution of $\pi = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$ upon inputs $(-, (T, (p_1, \ldots, p_\lambda)))$ and auxiliary input z given to Adv.

Functionality $\mathcal{F}_{\mathrm{OPM}}$

Let $m, \lambda \in \mathbb{N}$. Functionality $\mathcal{F}_{\mathrm{OPM}}$ sets the table $\mathcal{B}$ initially to the empty and proceeds as follows, running with clients C_T and C, server S and adversary Sim.

1. Upon receiving a message (text, T, m) from C_T, send $(\mathsf{preprocess}, |T|, m)$ to S and Sim, and record (text, T).
2. Upon receiving a message (query, p_i) from client C (for $i \in [\lambda]$), where message $(\mathsf{text}, \cdot)$ has been recorded and $|p_i| = m$, it checks if the table $\mathcal{B}$ already contains an entry of the form $(p_i, \cdot)$. If this is not the case then it picks the next available identifier id from $\{0,1\}^*$ and adds (p_i, id) to $\mathcal{B}$. It sends (query, C) to C_T and Sim.
   (a) Upon receiving $(\mathsf{approve}, C)$ from client C_T, read (p_i, id) from $\mathcal{B}$ and send $(\mathsf{query}, C, (i_1, \ldots, i_t), \mathsf{id})$ to server S, for all text positions $\{i_j\}_{j \in [t]}$ such that $T_{i_j} = p_i$. Otherwise, if no $(\mathsf{approve}, C)$ message has been received from C_T, send $\perp$ to C and abort.
   (b) Upon receiving $(\mathsf{approve}, C)$ from Sim, read (p_i, id) from $\mathcal{B}$ and send $(\mathsf{query}, p_i, (i_1, \ldots, i_t), \mathsf{id})$ to client C. Otherwise, send $\perp$ to client C.

Fig. 1. The outsourced pattern matching functionality

Definition 1 (Security of outsourced pattern matching). *We say that* π securely implements $\mathcal{F}_{\mathrm{OPM}}$*, if for any PPT real adversary* Adv *there exists a PPT simulator* Sim *such that for any tuple of inputs* $(T, (p_1, \ldots, p_\lambda))$ *and auxiliary input* z*,*

$$\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OPM}}, \mathsf{Sim}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa \in \mathbb{N}} \overset{c}{\approx} \{\mathbf{REAL}_{\pi, \mathsf{Adv}(z)}(\kappa, (-, T, (p_1, \ldots, p_\lambda)))\}_{\kappa \in \mathbb{N}}.$$

The schemes described in the next sections, implement the ideal functionality $\mathcal{F}_{\mathrm{OPM}}$ in the random oracle model.

3 A Scheme with Passive Security

In this section we present our implementation of the outsourced pattern matching functionality $\mathcal{F}_{\mathrm{OPM}}$ that is formalized in Fig. 1, and prove its security against semi-honest adversaries. A scheme with security against malicious adversaries is described in the full version of this paper [9], building upon the protocol in this section. Recall first that in the outsourced variant of the pattern matching problem, client C_T preprocesses the text T and then stores it on the server S in such a way that the preprocessed text can be used later to answer search queries submitted by client C. The challenge is to find a way to hide the text (in order to obtain privacy), while enabling the server to carry out searches on the hidden text whenever it is in possession of an appropriate trapdoor.

Protocol $\pi_{\mathsf{SH}} = (\pi_{\mathsf{Pre}}, \pi_{\mathsf{Query}}, \pi_{\mathsf{Opm}})$

Let $\kappa \in \mathbb{N}$ be the security parameter and let M, m, n, μ be integers, where for simplicity we assume that n is a multiple of $2m$. Further, let $\mathcal{H} : \{0,1\}^{\mu} \to \mathbb{Z}_M$ be a random oracle and $\mathsf{F} : \{0,1\}^{\kappa} \times \{0,1\}^{m} \to \{0,1\}^{\mu}$ be a PRF. Protocol π_{SH} involves a client C_T holding a text $T \in \{0,1\}^n$, a client C querying for patterns $p \in \{0,1\}^m$, and a server S. The interaction between the parties is specified below.

Setup phase, π_{Pre}**.** The protocol is invoked between client C_T and server S. Given input T and integer m, client C_T picks a random key $k \in \{0,1\}^{\kappa}$ and prepares first the text T for the packaging by writing it as

$$\begin{aligned} T' := (B_1, \ldots, B_u) = ((T[1], \ldots, T[2m]),& \\ (T[m+1], \ldots, T[3m]), \ldots, (T[n-2m+1], \ldots, T[n])),& \end{aligned}$$

where $u = n/m - 1$. Next, for each block B_b and each of the $m+1$ patterns $p \in \{0,1\}^m$ that appear in B_b we proceed as follows (suppose there are at most t matches of p in B_b).

1. Client C_T evaluates $R_p := \mathcal{H}(\mathsf{F}(k,p)||b)$, samples $a_1, \ldots, a_{t-1} \in \mathbb{Z}_M$ at random and then fixes a_t such that $a_t = R_p - \sum_{j=1}^{t-1} a_j \bmod M$.
2. Set $\widetilde{B}_b[v_j] = a_j$ for all $j \in [t]$ and $v_j \in [m+1]$. Note that here we denote by $\{v_j\}_{j \in [t]}$ ($v_j \in [m+1]$) the set of indexes corresponding to the positions where p occurs in B_b. Later in the proof we will be more precise and explicitly denote to which block v_j belongs by using explicitly the notation v_{j_b}.

Finally, we outsource the text $\widetilde{T} = (\widetilde{B}_1, \ldots, \widetilde{B}_u)$ to S.

Query phase, π_{Query}**.** Upon issuing a query $p \in \{0,1\}^m$ by client C, clients C_T and C engage in an execution of protocol π_{Query} which implements the oblivious PRF functionality $(k,p) \mapsto (-, \mathsf{F}(k,p))$. Upon completion, C learns $\mathsf{F}(k,p)$.

Oblivious pattern matching phase, π_{Opm}**.** This protocol is executed between server S (holding $\widetilde{T}$) and client C (holding $\mathsf{F}(k,p)$). Upon receiving $\mathsf{F}(k,p)$ from C, the server proceeds as follows for each block $\widetilde{B}_b$. It interprets $(\mathcal{H}(\mathsf{F}(k,p)||b), \widetilde{B}_b)$ as a subset sum instance and computes $\mathbf{s}$ as the solution of $\widetilde{B}_b \cdot \mathbf{s} = \mathcal{H}(\mathsf{F}(k,p)||b)$. Let $\{v_j\}_{j \in [t]}$ denote the set of indexes such that $\mathbf{s}[v_j] = 1$, then the server S returns the set of indexes $\{\varphi(b, v_j)\}_{b \in [u], j \in [t]}$ to the client C.

Fig. 2. Semi-honest outsourced pattern matching

We consider a new approach and reduce the pattern matching problem to the subset sum problem. Namely, consider a text T of length n, and assume we want to allow to search for patterns of length m. For some integer $M \in \mathbb{N}$, we assign to each distinct pattern p that appears in T a random element $R_p \in \mathbb{Z}_M$. Letting $\ell = n - m + 1$, the preprocessed text $\widetilde{T}$ is a vector in $\mathbb{Z}_M^{\ell}$ with elements specified as follows. For each pattern p that appears t times in T, we sample random values $a_1, \ldots, a_t \in \mathbb{Z}_M$ such that $R_p = \sum_{j=1}^{t} a_j$. Denote with $i_j \in [\ell]$ the jth position in T where p appears and set $\widetilde{T}[i_j] = a_j$. Notice that for each pattern p, there exists a vector $\mathbf{s} \in \{0,1\}^{\ell}$ such that $R_p = \widetilde{T} \cdot \mathbf{s}$. Hence, the positions in $\widetilde{T}$ where pattern p matches are identified by a vector $\mathbf{s}$ and can be viewed as the solution for the subset sum problem instance $(R_p, \widetilde{T})$.

Roughly, our protocol works as follows. During protocol π_{Pre}, we let the client C_T generate the preprocessed text $\widetilde{T}$ as described above, and send the result to the server S. Later, when a client C wants to learn at which positions a pattern p matches in the text, clients C and C_T run protocol π_{Query}; at the end of this protocol, C learns the trapdoor R_p corresponding to p. Hence, during π_{Opm}, client C sends this trapdoor to S, which can solve the subset sum problem instance $(R_p, \widetilde{T})$. The solution to this problem corresponds to the matches of p, which are forwarded to the client C. To avoid that C_T needs to store all trapdoors, we rely on a PRF to generate the trapdoors itself. More precisely, instead of sampling the trapdoors R_p uniformly at random, we set $R_p := \mathsf{F}(k, p)$, where F is a PRF. Thus, during the query phase C and C_T run an execution of an oblivious PRF protocol; where C learns the output of the PRF, i.e., the trapdoor R_p.

Efficiency. Although the protocol described above provides a first basic solution for the outsourced pattern matching, it suffers from a strong restriction as only very short texts are supported. (On the positive side, the above scheme does not rely on a random oracle.) The server S is asked to solve subset sum instances of the form $(\widetilde{T}, R_p)$, where $\widetilde{T}$ is a vector of length $\ell = n - m + 1$ with elements from $\mathbb{Z}_M$ for some integer M. To achieve correctness, we require that each subset sum instance has a unique solution with high probability. In order to satisfy this property, one needs to set the parameters such that the value $2^\ell/M$ is negligible. Fixing $M = 2^{\kappa+\ell}$ achieves a reasonable correctness level.

On the other hand, we need to let S solve subset sum instances efficiently. The hardness of subset sum depends on the ratio between ℓ and $\log M$, which is usually referred to as the *density* Δ of the subset sum instance. In particular both instances with $\Delta < 1/\ell$ (so called *low-density* instances) and $\Delta > \ell/\log^2 \ell$ (so called *high-density* instances) can be solved in polynomial time. Note that, however, the constraint on the ratio $2^\ell/M$ immediately rules out algorithms for high-density subset sum (e.g., algorithms based on dynamic programming, since they usually need to process a matrix of dimension M). On the other hand, for low-density instances, an easy calculation shows that $\ell + \kappa > \ell^2$, so that we need to choose κ, ℓ in such a way that

$$\ell < \frac{1}{2}\left(\sqrt{1+4\kappa} - 1\right). \tag{1}$$

The above analysis yields a value of $\ell \approx \sqrt{\kappa}$. This poses an inherent limitation on the length of the text. For instance, even using $\kappa \approx 10^4$ (yielding approximately subset sum elements of size 10 KByte) limits the length of the text to only 100 bits.

Packaging. To overcome this severe limitation, we partition the text into smaller pieces each of length $2m$, where each such piece is handled as a separate instance of the protocol. More specifically, for a text $T = (T[1], \ldots, T[n])$ let $(T[1], \ldots, T[2m])$, $(T[m+1], \ldots, T[3m]), \ldots$ be blocks, each of length $2m$, such that every two consecutive blocks overlap in m bits. Then, for each pattern p that appears in the text the client C_T computes an individual trapdoor for each block where the pattern p appears. In other words, suppose that pattern p appears in block B_b then we compute the trapdoor for this block (and pattern p) as $\mathcal{H}(\mathsf{F}(k,p)||b)$. Here, $\mathcal{H}$ is a cryptographic hash function that will be modeled as a random oracle in our proofs. Given the trapdoors, we apply the preprocessing algorithm to each block individually.

The sub-protocols π_{Query} and π_{Opm} work as described above with a small change. In π_{Query} client C learns the output of the PRF $\mathsf{F}(k,p)$ instead of the actual trapdoors and in π_{Opm} client C forwards directly the result $\mathsf{F}(k,p)$ to S. The server can then compute the actual trapdoor using the random oracle. This is needed to keep the communication complexity of the protocol low. Note that in this case if we let $\{v_{j_b}\}_{j_b \in [t_b]}$ be the set of indices corresponding to the positions where p occurs *in a given block* B_b, the server needs to map these positions to the corresponding positions in T (and this has to be done for each of the blocks where p matches). It is easy to see that such a mapping from a position v_{j_b} in block B_b to the corresponding position in the text T can be computed as $\varphi(b, v_j) = (b-1)m + v_j$. The entire protocol is shown in Fig. 2.

Note that now each of the preprocessed blocks $\widetilde{B}_b$ consist of $\ell = m+1$ elements in $\mathbb{Z}_M$. The advantage is that the blocks are reasonably short which yields subset sum instances of the form $(\widetilde{B}_b, R_p)$. Combined with Eq. (1) this yields a value of $\ell = 2m - m + 1 = m + 1 < \sqrt{\kappa}$, which is an upper bound on the length of the pattern (and not on the length of the text as before). By combining many blocks we can support texts of *any length* polynomial in the security parameter. Finally, we emphasize that the communication/computational complexities of π_{Query} depends on the underlying oblivious PRF evaluation. This in particular only depends on m (due to the algebraic structure of the [21] PRF). Using improved PRFs can further reduce the communication complexity. On the other hand, the communication complexity of π_{Opm} is dominated by the number of matches of p in T which is optimal.

We state the following result. The proof can be found in the full version [9].

Theorem 1. *Let $\kappa \in \mathbb{N}$ be the security parameter. For integers n, m we set $\lambda = \mathsf{poly}(\kappa), \mu = \mathsf{poly}(\kappa)$, $u = n/m - 1$, $\ell = (m+1)u$ and $M = 2^{m+\kappa+1}$. We furthermore require that κ is such that $2^{m+1}/M$ is negligible (in κ). Assume $\mathcal{H} : \{0,1\}^\mu \to \mathbb{Z}_M$ is a random oracle and $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^m \to \{0,1\}^\mu$ is a pseudorandom function. Then, protocol π_{SH} from Fig. 2 securely implements the $\mathcal{F}_{\mathrm{OPM}}$ functionality in the presence of semi-honest adversaries.*

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: From secrecy to soundness: Efficient verification via secure computation. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 152–163. Springer, Heidelberg (2010)
2. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012)

3. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011)
4. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
5. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: STOC, pp. 639–648 (1996)
6. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
7. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.-P., Stern, J.: Improved low-density subset sum algorithms. Computational Complexity 2, 111–128 (1992)
8. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 54–74. Springer, Heidelberg (2012)
9. Faust, S., Hazay, C., Venturi, D.: Outsourced pattern matching. Cryptology ePrint Archive, Report 2013/XX, `http://eprint.iacr.org/`
10. Frieze, A.M.: On the lagarias-odlyzko algorithm for the subset sum problem. SIAM J. Comput. 15(2), 536–539 (1986)
11. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178 (2009)
13. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. J. Cryptology 9(4), 199–216 (1996)
14. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
15. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. IACR Cryptology ePrint Archive, 2011:272 (2011)
16. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. J. ACM 32(1), 229–246 (1985)
17. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC, pp. 1219–1234 (2012)
18. Lyubashevsky, V., Palacio, A., Segev, G.: Public-key cryptographic primitives provably as secure as subset sum. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 382–400. Springer, Heidelberg (2010)
19. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: FOCS, pp. 80–91 (2003)
20. Mohassel, P.: Efficient and secure delegation of linear algebra. IACR Cryptology ePrint Archive 2011:605 (2011)
21. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: FOCS, pp. 458–467 (1997)
22. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 91–110. Springer, Heidelberg (2011)