Towards Secure Bioinformatics Services

M. Franz*, B. Deiseroth*, K. Hamacher+, S. Jha#, S. Katzenbeisser*, H. Schröder*

* Technische Universität Darmstadt, Security Engineering Group
+ Technische Universität Darmstadt, Computational Biology Group
# University of Wisconsin, Computer Sciences Department

Abstract. In this paper we show how privacy of genomic sequences can be protected while they are analyzed using Hidden Markov Models (HMM), which is commonly done in bioinformatics to detect certain non-beneficial patterns in the genome. Besides offering strong privacy guarantees, our solution also allows protecting the intellectual property of the parties involved, which makes the solution viable for implementation of secure bioinformatics services. In particular, we show how two mutually mistrusting parties can obliviously run the forward algorithm in a setup where one party knows a HMM and another party knows a genomic string; while the parties learn whether the model fits the genome, they neither have to disclose the parameterization of the model nor the sequence to each other. Despite the huge number of arithmetic operations required to solve the problem, we experimentally show that HMMs with sizes of practical importance can obliviously be evaluated using computational resources typically found in medical laboratories. As a central technical contribution, we give improved protocols for secure and numerically stable computations on non-integer values.

1 Introduction

It is commonly believed that advances in and economies of scale of biological sequencing will allow to sequence human genomes at low cost in the foreseeable future, effectively paving the route for personalized medicine [19, 20]. Genomic data will be used by healthcare providers to check for disease predispositions or drug intolerances of individual patients. It can be foreseen that a service-based industry will emerge, where special providers offer services to match genomic sequences against models of specific diseases, as well as offering personalized programs of drug application to such patients.

In this context two important security problems arise: First, genomic data must be considered extremely privacy sensitive and should thus be strongly protected from abuse. Second, mathematical models for diseases and the related genomic details are valuable intellectual property of the service provider, which is the basis of his business model. Enabling such bioinformatics services thus requires security mechanisms that can achieve both goals at the same time: privacy protection of genomic data and protection of the involved intellectual property.

So far, existing works considered basic bioinformatics algorithms that search for patterns (represented by regular expressions or substrings), perform sequence alignments or compute typical quantities, such as the edit distance (e.g. see [3, 12, 18]). In this paper we show how to evaluate much more complex probabilistic queries represented by Hidden Markov Models (HMMs) on genomic data in a secure way. HMMs are widely used in computational biology to detect locations with certain properties, e.g., disease related signals or general properties of important, disease related enzymes, such as kinases.

In particular we consider the following scenario: Party $\mathcal{A}$, who is a health care provider acting on behalf of a patient, has sequenced the patient's genome. $\mathcal{A}$ wants to interact with a provider $\mathcal{B}$, who offers a service to check parts of the genome against a model, encoded as HMM, for a specific disease. $\mathcal{A}$ wants to see "how good" the model fits the genome, thereby determining the likelihood of a disease predisposition, while being bound to preserve the patient's privacy. At the same time, party $\mathcal{B}$ does not want to disclose the HMM since the model itself is his business secret that distinguishes his service from other providers. In this paper we show how $\mathcal{A}$ and $\mathcal{B}$ can achieve both goals by providing a way to run the HMM *forward algorithm* [6] in an oblivious manner.

This requires techniques for efficiently and accurately performing Secure Multiparty Computation (SMC) on real values, since the forward algorithm operates on (sometimes extremely small) probabilities. A practical SMC-solution was first proposed in [9], which encodes (heavily quantized) real values in a logarithmic representation and provides protocols to obliviously perform all basic arithmetic operations on encryptions of such values. As a contribution of this paper we significantly improve performance of the primitives in [9] so that several hundred thousand of such arithmetic operations can be performed within a few minutes.

In summary, we make the following contributions in this paper:

- We improve the most complex operation (which is the addition) of the framework presented in [9]; while the original solution required $\mathcal{O}(|\mathcal{T}|)$ computation, our improved version requires only $\mathcal{O}(\sqrt{|\mathcal{T}|})$ work.
- We further give a private implementation of the forward algorithm which is commonly used to process and analyze genomic material and protein sequences. The resulting protocol allows to protect both the privacy of genomic data and the intellectual property of the service provider (i.e., the HMM).
- We implemented the algorithm in the widely used HMMER [7] framework and tested it on realistic models and sequences. We show that the developed protocols allow to analyze medium-size models on standard computing equipment in a couple of minutes, despite performing about 300.000 arithmetic operations on small encrypted probabilities.

2 Related Work

Several constructions for Secure Multiparty Computation [21] are known in the literature. Various approaches for securely evaluating a function have been developed for different function representations, namely combinatorial circuits [10, 11], Ordered Binary Decision Diagrams [13], branching programs [15], or one-dimensional look-up tables [14]. While these methods target computations performed over the integers only, extensions were proposed that can handle rational numbers [8], real values in fixed point notation [4] and real values in logarithmic notation [9].

Some works deal with the secure analysis of genomic sequences: [3] considers the problem of securely computing the edit distance between two strings; [12] presents a secure algorithm to perform sequence alignments using the Smith-Waterman algorithm; finally [18] proposes an algorithm that allows to run arbitrary queries, formulated as regular expressions, on genomic sequences in an oblivious way. Some papers deal with the problem of securely evaluating HMMs [17, 16]. However, neither of these protocols comes with a satisfactory security proof and sufficient experimental analysis to assure the accuracy of the results.

3 Efficient Computations on Encrypted Non-Integer Values

As a central cryptographic tool, we use a semantically secure additively homomorphic public-key encryption scheme introduced by Damgård, Geisler and Krøigaard (DGK) in [5]. In DGK a message $m \in \mathbb{Z}_u$ is encrypted by computing $c = g^m h^r \bmod n$, where n is a RSA-modulus, u is a prime number and r is a randomly chosen integer. In our application u is from a very small range, which results in a very small plaintext space $\mathbb{Z}_u$. For our construction it will be essential that the plaintext space is a small finite field, since this will allow us to perform very efficient operations (such as polynomial interpolation in $\mathbb{Z}_u$ or efficient decryption). Thus we will assume that the prime number u is chosen only slightly larger than the values which occur during the computations (for typical applications values u with bitlength 10-20 bits will be sufficient). In the sequel we will denote a DGK encrypted value m by $[m]$.

Secure computations on non-integer values. In [9], a framework was presented which allows secure computations on non-integer numbers. Rather than using a fixed point representation, the values are approximated using a logarithmic representation. This promises a constant relative representation error both for very small and for very large values. Each non-integer value $v \in R \subset \mathbb{R}$ is represented as a triplet (ρ, σ, τ), where ρ is a flag indicating whether v is equal to zero, σ stores the sign of v and $\tau = \lceil -S \cdot \log_B(\frac{|v|}{C}) \rfloor$ for positive parameters B, C, S. The framework provides protocols **LSUM**, **LSUB**, **LDIV**, and **LPROD**, which allow two parties to obliviously perform the four basic arithmetic operations on triplets $([\rho], [\sigma], [\tau])$ which have been encrypted using a semantically secure homomorphic cryptosystem. It can easily be seen that a product (**LPROD**) of such encrypted numbers x, y can be computed in a straightforward manner using the homomorphic properties of the encryption, by observing that $\tau_{xy} = \tau_x + \tau_y$. Unfortunately, the operation **LSUM** is more involved. To this end, the parties have to compute $\tau_{x+y} = \tau_y - S \cdot \lceil \log_B(1 + B^{(\tau_x - \tau_y)/S}) \rfloor$. Computing τ_{x+y} from τ_x and τ_y using standard methods from Secure Multiparty Computation is inefficient. Therefore, [9] uses an oblivious table lookup to obtain the value τ_{x+y}, so that none of the parties is able to learn anything about the value which was queried, and in turn, which value was retrieved by the oblivious look-up operation.

In the remainder of this section we will show how this table look-up operation can significantly be improved in order to obtain a more efficient **LSUM** implementation. In particular, we describe a two-party protocol which allows to perform an oblivious table lookup for some value $x \in X$, where $X = [x_l; x_u]$ is some interval with bounds $x_l, x_u \in \mathbb{Z}$. The table will be denoted by $\mathcal{T} = (x_i, f(x_i))_{x_i \in X}$; we use $\mathcal{T}(x)$ to denote the entry of the table $\mathcal{T}$ at position x. The construction is given in the well known two party scenario, where one party $\mathcal{A}$ holds the private key for some homomorphic

x	$f(x)$
x_1	$f(x_1)$
x_2	$f(x_2)$
$\vdots$	$\vdots$
$x_{\vert\mathcal{T}\vert}$	$f(x_{\vert\mathcal{T}\vert})$

Table 1. Table lookup in [9].

$y \setminus z$	0	1	$\dots$	$k-1$
0	$f(x_1)$	$f(x_2)$	$\dots$	$f(x_k)$
1	$f(x_{k+1})$	$f(x_{k+2})$	$\dots$	$f(x_{2k})$
$\vdots$	$\vdots$			
$k-1$	$f(x_{\vert\mathcal{T}\vert-k+1})$	$f(x_{\vert\mathcal{T}\vert-k+2})$	$\dots$	$f(x_{\vert\mathcal{T}\vert})$

Table 2. New table lookup.

encryption scheme, and another party $\mathcal{B}$ holds an encryption $[x]$ of a value x and learns an encryption $[\mathcal{T}(x)]$ of $\mathcal{T}(x)$, but neither x nor the plain table entry $\mathcal{T}(x)$.

Efficient Private Function Evaluation. Creating and transmitting the full table used in a **LSUM** is rather costly. Therefore, the main idea is to transform the lookup in a large table of size $|\mathcal{T}|$ into two lookups in smaller tables of size $k := \sqrt{|\mathcal{T}|}$ and one evaluation of a polynomial. This is, from a setting as depicted in Table 1, we go to a setting as depicted in Table 2. In the remainder of this section we assume that x is a positive value (this can always be achieved by shifting the interval $X = [x_l; x_u]$ to $X' = [0; x_u - x_l]$ and changing Table 1 accordingly). For simplicity, we will further assume that the table $\mathcal{T}$ has $2^{2\ell}$ entries, thus $k = \sqrt{|\mathcal{T}|} = 2^\ell$.

We first split up x into two values y and z which consist of the $\ell/2$ most significant resp. least significant bits of x, i.e. $x = y \cdot k + z$. Now, we perform two simultaneous table lookups. In the first lookup, the value z is mapped to a value $\tilde{x} \in \{\tilde{x}_0, \dots, \tilde{x}_{k-1}\}$, where the values $\tilde{x}_1, \dots, \tilde{x}_{k-1}$ are chosen at random from $\mathbb{Z}_u$. Next, polynomials P_i are generated in a way that on the evaluation points $\tilde{x}_1, \dots, \tilde{x}_{k-1}$ the polynomials take on values $f(x_i)$, i.e. $P_y(\tilde{x}_z) = f(y \cdot k + z)$. In the second lookup, the value y is used to select the polynomial P_y which is finally evaluated on $\tilde{x}$ to obtain the result $f(x) = f(y \cdot k + z) = P_y(\tilde{x})$.
Thus, the full protocol consists of the following steps:

1. (Offline): Prepare representation as polynomials: First we choose k random values $\tilde{x}_0, \dots, \tilde{x}_{k-1}$. Next, using Newton-interpolation, we compute k polynomials $P_y(\tilde{x}_z) = f(y \cdot k + z)$ such that $P_y(\tilde{x}_z) = f(y \cdot k + z)$.
2. Extract the bits of $[x]$ to obtain values $[y], [z]$, such that $x = y \cdot k + z$.
3. Transfer value a $\tilde{x}$ and a polynomial P with tables of size $k \approx \sqrt{|\mathcal{T}|}$: For this we can use the table-lookup as presented in [9] or use oblivious transfer (OT). First, use an oblivious table look-up to obtain the value $\tilde{x}_z$; next, run the protocol a second time to obtain the polynomial P_y.
4. Reconstruct the value $[\mathcal{T}(x)]$ by evaluating the polynomial P_y on the value $\tilde{x}_z$.

For the full protocol, a detailled description of all steps and a security proof we refer the reader to the full version of this paper.

4 Secure Bioinformatics

In this section we describe how the computational framework presented in Section 3 can be applied to algorithms that securely analyze genomic and gene product related sequences by using Hidden Markov Models (HMM). HMMs are probabilistic methods that model a stream of symbols by a Markov process, which is driven by "hidden" states of the process, unobservable to an outsider. In different states the dynamics of the process differs and thus the statistics of emitted symbols varies with time. States in bioinformatics applications can be indicators for particular disease related properties of sequences that code for e.g. non-beneficially mutated proteins or sites of potential post-translationally modifications.

Hidden Markov Models. A Hidden Markov Model $\lambda = (A, B, \pi)$ is characterized by the following elements:

- Each HMM contains a set S of N hidden states: $S = \{S_1, S_2, \ldots, S_N\}$.
- A set V of M distinct observation symbols per state (the output alphabet): $V = \{v_1, v_2, \ldots, v_M\}$.
- The state transition probability matrix $A = \{a_{ij}\}$, where a_{ij} is the probability of moving from state S_i to state S_j: $a_{ij} = \mathrm{Prob}[q_{t+1} = S_j \mid q_t = S_i]$ for $1 \leq i \leq N$ and $1 \leq j \leq N$, with proper normalization $\forall_{1\leq i\leq N} \sum_j a_{ij} = 1$.
- The emission probabilities $B = \{b_j(v_k)\}$ in state S_j, where $b_j(v_k)$ is the probability of emitting symbol v_k at state S_j: $b_j(k) = \mathrm{Prob}[v_k \text{ at } t \mid q_t = S_j]$ for $1 \leq j \leq N$ and $1 \leq k \leq M$.
- The initial state distribution $\pi = \{\pi_i\}$, where π_i is the probability that the start state is S_i: $\pi_i = \mathrm{Prob}[q_1 = S_i]$ for $1 \leq i \leq N$.

In typical bioinformatics applications (such as the one discussed in the introduction) a fundamental problem arises: Given a Hidden Markov Model λ and an observed sequence $O = o_1 o_2 \ldots o_T$, compute the probability $\mathrm{Prob}[O \mid \lambda]$ that the HMM can generate the sequence. This value indicates the significance of the observation. This problem is commonly solved by applying the forward algorithm.

Secure Forward Algorithm. We consider the following two-party scenario. Party $\mathcal{A}$ knows a genomic sequence O, while party $\mathcal{B}$ commands over a specific HMM λ. $\mathcal{A}$ could be a health care provider, who has sequenced a patient's genome O, but wishes to preserve the privacy of the patient. $\mathcal{B}$ is a drug company or some bioinformatics institute, which wants to preserve its intellectual property contained within the parameterization of the HMM. Both parties are interested to learn how good the model fits to the genome O by running the forward algorithm, whereas neither party wants to share its input with each other. The overall probability reported by the algorithm can be, for example, the confidence of $\mathcal{B}$ that $\mathcal{A}$'s patient develops a particular disease.

To compute the probability $\mathrm{Prob}[O \mid \lambda]$, we can employ the forward algorithm. Consider the so-called forward variable $\alpha_t(i) = \mathrm{Prob}[o_1, \ldots, o_t, q_t = S_i \mid \lambda]$ which indicates how likely it is to end up in state S_i after processing t steps, assuming that $o_1, \ldots, o_t$ have been emitted. For $\alpha_1(i)$ we have $\alpha_1(i) = \pi_i b_i(o_i)$. Given $\alpha_t(i)$, the probabilities $\alpha_{t+1}(i)$ can be computed inductively by $\alpha_{t+1}(i) = b_i(o_{t+1}) \cdot \sum_{j=1}^{N} \alpha_t(j) \cdot a_{ji}$. Finally, we have $\mathrm{Prob}[O \mid \lambda] = \sum_{i=1}^{N} \alpha_T(i)$.

A full description of the realization of the forward algorithm using the framework of Section 3 can be found in Protocol 1. Note that in the protocol for clearness of presentation we omit to explicitly label encoded elements: all values $a_{ij}, b_i(v_j), \pi_i$ and α_i should be read as encoded values according to Section 3. In the initialization step, party $\mathcal{A}$ provides party $\mathcal{B}$ with an encrypted version of the sequence, where each symbol of O is encoded as a binary vector of length M (the position of the one in the vector encodes

Protocol 1 Secure Forward Algorithm

```
Input: Party $\mathcal{A}$: Sequence $O = o_1 o_2 \dots o_T$
       Party $\mathcal{B}$: HMM $\lambda = (A, B, \pi)$
Output: $\mathrm{Prob}[O \mid \lambda]$
1: Initialization:
   Party $\mathcal{A}$:
   For each $o_i$ prepare a vector $\Theta_i = \{\theta_{i1}, \dots, \theta_{iM}\}$ in a way that $\theta_{ij} = 1$ if $v_j = o_i$ and $\theta_{ij} = 0$ otherwise.
   Encrypt $\Theta_i$ component wise and send $[\Theta_i]$ to party $\mathcal{B}$
2: Party $\mathcal{B}$:
   Compute emission probabilities:
   for $i = 1$ to $N$
     for $j = 1$ to $T$
       $[\rho_{b_i(o_j)}] = \prod_{k=1}^{M} [\theta_{ik}]^{\rho_{b_i(v_k)}}$
       $[\tau_{b_i(o_j)}] = \prod_{j=k}^{M} [\theta_{ik}]^{\tau_{b_i(v_k)}}$
       $[b_i(o_j)] := ([\rho_{b_i(o_j)}], [\tau_{b_i(o_j)}])$
     end
   end
3: Party $\mathcal{B}$:
   for $i = 1$ to $N$
     $[\alpha_1(i)] =$ LPROD$([\pi_i], [b_i(o_1)])$
   end
4: Induction:
   for $t = 1$ to $T - 1$
     for $i = 1$ to $N$
       $[\Sigma_1^1] =$ LPROD$([\alpha_t(1)], [a_{1i}])$
       for $j = 2$ to $N$
         tmp = LPROD$([\alpha_t(j)], [a_{ji})]$
         $[\Sigma_1^j] =$ LSUM$([\Sigma_1^{j-1}], )$
       end
       $[\alpha_{t+1}(i)] =$ LPROD$([b_i(o_{t+1})], [\Sigma_1^N])$
     end
   end
5: Termination:
   $[\Sigma_1^2] =$ LSUM$(\alpha_T(1), \alpha_T(2))$
   for $i = 3$ to $N$
     LSUM$([\Sigma_1^{i-1}], [\alpha_T(i)])$
   end
   return $[\mathrm{Prob}[O \mid \lambda]] := [\Sigma_1^N]$
```

the symbol). This allows party $\mathcal{B}$ to easily compute encryptions $[b_i(o_j)]$ of the emission probabilities by using the homomorphic properties of the Paillier cryptosystem in step 2. In addition, party $\mathcal{B}$ initializes the values $\alpha_1(i)$ as the product of the probabilities π_i and the emission probabilities $b_i(o_1)$ for the first observation symbol. In step 3 the parties compute interactively the forward-variables $\alpha_t(i)$, and in step 4 the result of the forward algorithm $[\mathrm{Prob}[O \mid \lambda]]$, which can be decrypted by $\mathcal{A}$.

5 Implementation and Experimental Results

We have implemented the optimized framework for performing secure computations on non-integer values as well as the secure forward algorithm in C++ using the GNU GMP library version 5.0.1. Tests were run on a computer with a 2.1 GHz 8 core processor and 4GB of RAM running Ubuntu version 9.04. The parties $\mathcal{A}$ and $\mathcal{B}$ were implemented as two multi-threaded entities of the same program. Therefore our tests do not include network latency.

*Complexity of **LSUM**.* We have implemented the **LSUM** operation using the optimizations described in Section 3. We compare our results to those of [9]. Figure 1 depicts the time complexity of the protocol of [9] and our optimized version. Both programs were run for different table sizes $|\mathcal{T}|$ in the range between $|\mathcal{T}| = 1.000$ and $|\mathcal{T}| = 20.000$, the implementation by [9] was allowed to reuse tables 150 times while our construction reused each set of polynomials 10 times. To allow for a fair comparison, we restricted both implementations to run only on one core of the processor (thus, no parallelization

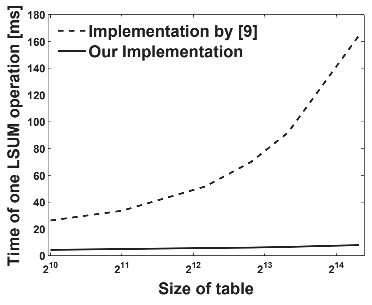

Fig. 1. Performance of the **LSUM** operation depending on the table size $|\mathcal{T}|$.

	Small	**Medium**	**Large**
Computation	33	499	632
Communication	69.6	964.3	1472.3
LSUM	14.161	162.491	225.984
LPROD	25.971	297.929	414.337

Fig. 2. Computational complexity (seconds), Communication complexity (MB), operation count for **LSUM** and **LPROD** of the forward algorithm for different model sizes.

was allowed in the tests). Figure 1 depicts the results: The y-axis shows the computational complexity in milliseconds (wall clock time) required to perform one **LSUM**, for different table sizes (x-axis, logarithmic scale). While the solution of [9] (dotted line) grows linear in $|\mathcal{T}|$ it can be seen that our improved solution grows moderately from 4.41ms for $|\mathcal{T}| = 1000$ to 8.64ms for $|\mathcal{T}| = 20.000$.

Complexity of private HMM analysis. In order to demonstrate the practicality of the secure forward algorithm developed in Section 4), we implemented this algorithm in HMMER [1], version 2.3.2, which is widely used in the bioinformatics community to perform analysis of DNA and protein sequences. Real values were encoded and encrypted as described in Section 3, while the **LSUM** operation was realized by using our optimized construction.

We tested our implementation with several HMMs from the PFAM database [2]. Among them are models which are relevant for identification of protein domains, that play crucial roles in oncogenic transformations and other diseases connected to signaling in cells. In particular, we chose HMMs of three different sizes: A small model (SH3_1, PF00018) of length 48, a medium model (Ras, PF00071) of length 162 and a large model (BID, PF06393) with 196 states. For the small, medium and large models we chose tables of size 1400, 3520 and 4600, respectively, in the **LSUM** operation. We experimentally chose the parameters of the number representation in a way that minimized the overall quantization error.

We measure the computational complexity of Protocols 1. The first row of Table 2 depicts the average runtime (wall clock time) of a single run of the forward algorithm. Note that in these tests we allowed parallel computations on multiple cores of the processor (e.g., we allowed to parallelize computation of the polynomials, run multiple **LSUM** operations in parallel, etc). Even though we did not fully optimize our programs at this point (only 3 out of 8 cores were used on average), we believe that the results are nevertheless insightful: For example, running the forward algorithm on the medium sized model requires approximately 8 minutes, despite performing 297.929 invocations of **LPROD** and 162.491 invocations of **LSUM**.

The communication complexity measures the traffic between the two parties. This value mainly depends on the size of the RSA modulus n, which was set to 1024 bits.

The keysize for the garbled circuits was set to 80 bit. The second row of Table 2 depicts the communication complexity.

Acknowledgements. Kay Hamacher gratefully acknowledges financial support by Fonds der chemischen Industrie. This work was supported by the German Research Foundation (DFG) and the Center for Advanced Security Research Darmstadt (CASED).

References

1. Hmmer, biosequence analysis using profile hidden markov models. http://hmmer.wustl.edu/.
2. Pfam version 24.0. http://pfam.sanger.ac.uk.
3. Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. Secure and private sequence comparisons. In *ACM Workshop on Privacy in the Electronic Society*, pages 39–44, 2003.
4. O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography*, 2010.
5. I. Damgård, M. Geisler, and M. Krøigaard. Efficient and Secure Comparison for On-Line Auctions. In *ACSIP 2007*, volume 4586 of *LNCS*, 2007.
6. R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis – Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
7. S R Eddy. Profile hidden markov models. *Bioinformatics*, 14(9):755–63, Jan 1998.
8. Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers. Cryptocomputing with rationals. In *Financial Cryptography*, pages 136–146, 2002.
9. M. Franz, B. Deiseroth, K. Hamacher, S. Katzenbeisser, S. Jha, and H. Schroeder. Secure computations on real-valued signals. In *Proceedings of the IEEE Workshop on Information Forensics and Security - WIFS'10*, 2010.
10. O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *ACM Symposium on Theory of Computing — STOC '87*, pages 218–229. ACM, May 25-27, 1987.
11. M. Jacobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT'00*, volume 1976 of *LNCS*, pages 162–177. Springer-Verlag, 2000.
12. S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy*, pages 216–230, 2008.
13. L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM CCS*, pages 410–420, 2006.
14. M. Naor and K. Nissim. Communication complexity and secure function evaluation. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(062), 2001.
15. M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *ACM Symposium on Theory of Computing*, pages 590–599, 2001.
16. Huseyin Polat, Wenliang Du, Sahin Renckes, and Yusuf Oysal. Private predictions on hidden markov models. *Artif. Intell. Rev.*, 34(1):53–72, 2010.
17. P. Smaragdis and M. Shashanka. A framwork for secure speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 15(4):1404–1413, 2007.
18. J. Ramón Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *ACM CCS*, pages 519–528, 2007.
19. Laura J. van 't Veer and Rene Bernards. Enabling personalized cancer medicine through analysis of gene-expression patterns. *Nature*, 452(7187):564–570, Apr 2008.
20. M. West, G. S. Ginsburg, A. T. Huang, and J. R. Nevins. Embracing the complexity of genomic data for personalized medicine. *Genome Research*, 16(5):559–566, 2006.
21. A. C.-C. Yao. Protocols for Secure Computations (Extended Abstract). In *Annual Symposium on Foundations of Computer Science — FOCS '82*, pages 160–164. IEEE, 1982.