Alon Rosen

Cryptography for ML

What have the Romans ever done for us?[*]

Cryptography goes significantly beyond protecting the secrecy of transmitted messages. Tools such as *fully homomorphic encryption* (FHE) and *two-party computation* (2PC) enable a client holding an input $\mathbf{x}$ to learn precisely $f_{\boldsymbol{\theta}}(\mathbf{x})$ without revealing $\mathbf{x}$ to the server, which holds $f_{\boldsymbol{\theta}}$.

Whether the adversary controls the client or the server, it learns *nothing* beyond what is intentionally revealed by the prescribed input–output behavior.

The price for such strong security is a hefty performance penalty. Modern ML applications include some of the heaviest computations performed and can afford hardly any overhead beyond the cost of computing $f_{\boldsymbol{\theta}}(\mathbf{x})$ itself.

This calls for cryptographic solutions tailored to modern ML hardware and workloads, minimizing overhead beyond plain computation. *Private vector search*, a prevalent functionality in modern ML systems, is an important example: its large linear-algebra core can be heavily optimized, leaving only a much smaller non-linear selection step for generic secure computation.

The main points of this chapter are:

- FHE: low communication; noise and parameters grow with multiplicative depth.
- 2PC: cheap local computation; communication and rounds scale with the circuit.
- Private vector search: specialized protocols exploit linear-algebra structure for low overhead.

[*] Monty Python, *Life of Brian*.

1 Encryption

In *secret-key encryption*, a sender wishes to securely communicate a message m to a receiver, under the assumption that both parties agree on a secret random key in advance.

Definition 1 (Syntax)**.** *A secret-key encryption scheme is* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *with* $k \leftarrow \mathrm{Gen}$ *and* $c \leftarrow \mathrm{Enc}_k(m)$*. It is correct if* $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$ *for every* m*.*

The syntax alone can be satisfied by sending m as the ciphertext. But then all information about m is revealed. The challenge is to convincingly hide m, which calls for a definition of security. The definition is *maximalistic*: security should hide *all* efficiently accessible information, even when the adversary can request encryptions of messages of its choice.

Definition 2 (Security)**.** *An encryption scheme is* (t, ε)*-secure if, for every pair of equal-length messages* m_0, m_1*, over* $k \leftarrow \mathrm{Gen}$*,*

$$\mathrm{Enc}_k(m_0) \approx_c \mathrm{Enc}_k(m_1).$$

Here $X \approx_c Y$ denotes *computational indistinguishability*: every randomized algorithm A running in time at most t, with oracle access to $\mathrm{Enc}_k(\cdot)$, satisfies

$$\left|\Pr[A^{\mathrm{Enc}_k(\cdot)}(X) = 1] - \Pr[A^{\mathrm{Enc}_k(\cdot)}(Y) = 1]\right| \le \varepsilon.$$

Why does this guarantee security? If any efficiently computable property of m could be learned from $\mathrm{Enc}_k(m)$, choosing m_0, m_1 that differ on this property would distinguish their encryptions.

Security alone can be satisfied by sending nothing. But then there is no correctness. Remarkably, Definition 1 and Definition 2 can be achieved simultaneously.

Proposition 1 (LWE encryption)**.** *Assuming LWE is hard, secure encryption exists.*

Proof. Fix plaintext modulus p, let $q \gg p$, and write $\Delta = \lfloor q/p \rfloor$. Suppose for simplicity that the error e always satisfies $|e| < \Delta/2$. The key is a uniformly random $\mathbf{s} \leftarrow \mathbb{Z}_q^n$. To encrypt $m \in \mathbb{Z}_p$, sample uniform $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and an error e, and output

$$\mathrm{Enc}_{\mathbf{s}}(m) = \big(\mathbf{a},\, \mathbf{a}^\top \mathbf{s} + e + \Delta m\big) \pmod q.$$

Decryption subtracts $\mathbf{a}^\top \mathbf{s}$, divides by Δ, and rounds to the nearest element of $\mathbb{Z}_p$. Specifically, $\mathrm{Dec}_{\mathbf{s}}(\mathbf{a}, b) = \mathrm{round}((b - \mathbf{a}^\top \mathbf{s})/\Delta) \bmod p = \mathrm{round}(m + e/\Delta) \bmod p = m$.

Suppose an adversary could distinguish $\mathrm{Enc}_{\mathbf{s}}(m_0)$ from $\mathrm{Enc}_{\mathbf{s}}(m_1)$. We could then distinguish LWE samples from uniform pairs. Given a pair $(\mathbf{a}, z)$, answer a request for an encryption of m with $(\mathbf{a}, z + \Delta m)$. If the pairs are LWE samples, these are exactly real encryptions. If they are uniform pairs $(\mathbf{u}, v) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, then $v + \Delta m$ is uniform for every m, so the adversary's entire view is independent of whether the distinguished ciphertext encrypts m_0 or m_1. Thus the adversary has no advantage in the uniform case, and hence negligible advantage in the LWE case. □

Without a computational restriction on the adversary, the length of the secret key must grow with the length of the message, so one short key cannot protect arbitrarily many encryptions.

Encryption in the scheme above is randomized, as it must be. If Enc_k were deterministic, the adversary could request $\mathrm{Enc}_k(m_0)$ and distinguish it from $\mathrm{Enc}_k(m_1)$ by testing equality.

2 Homomorphic encryption

Apart from encryption, *what has cryptography has ever done for us*? Well, what about *homomorphic encryption*? Suppose $c = \mathrm{Enc}_k(x)$ encrypts a value x. For a known computation C, $\mathrm{Eval}(C, c)$ produces another ciphertext. Correctness requires

$$\mathrm{Dec}_k(\mathrm{Eval}(C, \mathrm{Enc}_k(x))) = C(x).$$

If Eval supports all polynomial-size circuits C of multiplicative depth at most a bound L fixed in advance, the scheme is *leveled homomorphic*. If it supports all polynomial-size circuits with no a-priori depth bound, the scheme is *fully homomorphic*.

The applicability to private inference is immediate: a client holding x sends $\mathrm{Enc}_k(x)$ to a server holding $f_{\boldsymbol{\theta}}$; the server returns $\mathrm{Eval}(f_{\boldsymbol{\theta}}, \mathrm{Enc}_k(x))$, and the client decrypts to obtain $f_{\boldsymbol{\theta}}(x)$. A subtle issue is that $\mathrm{Eval}(f_{\boldsymbol{\theta}}, \mathrm{Enc}_k(x))$, together with k, may reveal more about $f_{\boldsymbol{\theta}}$ than $f_{\boldsymbol{\theta}}(x)$; we will ignore this *circuit privacy* concern, but it can be handled.

Security is preserved under evaluation. Since Eval is efficient, any efficient algorithm that distinguishes evaluated ciphertexts could be composed with Eval to distinguish the original ciphertexts.

Theorem 1 (Leveled HE)**.** *Assuming LWE is hard, secure leveled homomorphic encryption exists.*

The theorem follows from the two propositions below, which we explain for the LWE encryption of Proposition 1. Addition evaluates known linear functions. Multiplication is also possible, but worsens the ciphertext parameters and increases the error.

Proposition 2 (Homomorphic addition)**.** *Known linear functions of encrypted values can be evaluated homomorphically as long as the accumulated error remains below the decoding threshold.*

Proof. If $c_i = (\mathbf{a}_i, b_i)$, where $b_i = \mathbf{a}_i^\top \mathbf{s} + e_i + \Delta m_i \pmod q$, encrypts m_i, then for coefficients α_i,

$$\sum_i \alpha_i b_i - \left(\sum_i \alpha_i \mathbf{a}_i\right)^\top \mathbf{s} = \Delta \sum_i \alpha_i m_i + \sum_i \alpha_i e_i.$$

Thus addition of ciphertexts adds plaintexts, while multiplying a ciphertext by a known constant α multiplies both the plaintext and its error by α. Correctness lasts only while $|\sum_i \alpha_i e_i| < \Delta/2$. □

Proposition 3 (Homomorphic multiplication)**.** *Two encrypted values can be multiplied homomorphically, provided the parameters leave enough room for the increased error.*

Proof. Multiplying the two decryption relations and rescaling by Δ gives

$$\Delta m_1 m_2 + m_1 e_2 + m_2 e_1 + \frac{e_1 e_2}{\Delta}.$$

The product is now quadratic in the secret key and naturally produces a ciphertext with $O(n^2)$ coefficients. A standard technique called *relinearization* uses auxiliary evaluation information to rewrite these quadratic secret-key terms as linear ones, returning to the usual $O(n)$-size ciphertext form but adding a further error. Thus one multiplication replaces the old errors by roughly

$$m_1 e_2 + m_2 e_1 + \frac{e_1 e_2}{\Delta} + e_{\mathrm{relin}},$$

and repeated multiplications rapidly consume the $\Delta/2$ decoding margin. The modulus and related parameters must therefore grow with the allowed multiplicative depth. □

A leveled scheme chooses parameters large enough for a prescribed multiplicative depth. To remove the depth bound, before the error becomes too large the scheme *bootstraps*: it homomorphically evaluates its own decryption procedure and thereby refreshes the ciphertext.

A ciphertext contains $O(n)$ elements of $\mathbb{Z}_q$, hence $O(n \log q)$ bits. With n_{in} encrypted inputs and n_{out} encrypted outputs, the online communication is $O((n_{\text{in}} + n_{\text{out}}) n \log q)$ bits, independent of the size of the evaluated circuit; after the upload, evaluation is noninteractive. Packing B plaintext values into one ciphertext reduces the amortized communication per value by a factor B. The evaluation information is polynomial-size and can be reused across evaluations.

Computation is much heavier. Homomorphic addition costs $O(n)$ modular operations. A multiplication forms $O(n^2)$ coefficient products and then relinearizes, so it costs $\text{poly}(n, \log q)$ modular operations and increases the error. Thus a circuit with s gates and multiplicative depth d can be evaluated in $s \cdot \text{poly}(n, \log q)$ time, with q and, when needed, n chosen larger as d grows.

Bootstrapping costs $\text{poly}(n, \log q)$ operations per refresh and polynomial-size evaluation information, so it can dominate time and memory. Batching, GPUs, and specialized hardware improve constants through parallelism, but not these asymptotic costs.

3 Secure two-party computation

All right, I'll grant you encryption, homomorphic encryption, authentication, and even obfuscation. But *what has cryptography ever done for us*?

In *two-party computation*, two parties compute jointly while keeping both inputs hidden from each other. A two-party functionality F maps inputs (x, y) to prescribed outputs

$$F(x, y) = \big(F_1(x, y), F_2(x, y)\big).$$

In the *real world*, the parties run a protocol Π. In the *ideal world*, they give their inputs to a trusted party, which evaluates F and returns only the prescribed outputs.

Definition 3 (Simulation security for two-party computation)**.** *A protocol Π securely realizes F if, for every probabilistic polynomial-time adversary A corrupting either party, there is a probabilistic polynomial-time simulator S such that, for every x, y and auxiliary input z,*

$$\text{REAL}_{\Pi,A}(x, y; z) \approx_c \text{IDEAL}_{F,S}(x, y; z).$$

The simulator must reproduce the adversary's real view using only what the ideal functionality is allowed to reveal. For private inference we may take

$$F(\mathbf{x}, \boldsymbol{\theta}) = \big(f_{\boldsymbol{\theta}}(\mathbf{x}), \perp\big).$$

Thus the protocol should leak nothing beyond what follows from the permitted input–output behavior itself. The point is that a real-world adversary should have no significantly greater advantage in attacking the protocol than an ideal adversary. In the ideal world, *there is no attack*: what the ideal adversary learns is precisely what is given by the output of F, *and nothing else*.

Here we assume adversaries are *semi-honest*: they follow the protocol's prescribed instructions but try to learn from everything they see. In the next chapter we will see how to force parties to follow the protocol, yielding security against *malicious* adversaries.

Theorem 2 (2PC from LWE)**.** *Assuming LWE is hard, every polynomial-time two-party functionality has a computationally secure protocol against semi-honest adversaries.*

Proof. It suffices to construct *1-out-of-2 oblivious transfer*: below, two OTs suffice to evaluate one NAND gate on shared bits, and NAND gates can express every Boolean circuit. In 1-out-of-2 OT, the sender holds (m_0, m_1), the receiver holds a choice bit b, and the functionality is

$$\mathcal{F}_{\mathrm{OT}}\big((m_0, m_1), b\big) = \big(\perp, m_b\big).$$

The additive LWE scheme gives OT as follows. The receiver chooses a key k, sends $c = \mathrm{Enc}_k(b)$ together with a collection of encryptions of 0, and the sender forms from a random subset a fresh encryption z of 0. It returns

$$d = z + (m_1 - m_0)c + \Delta m_0 = \mathrm{Enc}_k(m_0 + (m_1 - m_0)b) = \mathrm{Enc}_k(m_b).$$

The receiver decrypts d. The random subset rerandomizes the ciphertext, while encryption security hides b from the sender.

To compute NAND securely, parties A and B randomly share x and y as

$$x = x_A \oplus x_B, \qquad y = y_A \oplus y_B.$$

Then

$$xy = x_A y_A \oplus x_A y_B \oplus x_B y_A \oplus x_B y_B.$$

Each party computes its term $x_A y_A$ and $x_B y_B$, respectively. For $x_A y_B$, party A samples r and sends OT messages $(r, r \oplus x_A)$, while B chooses y_B; their outputs r and $r \oplus x_A y_B$ share $x_A y_B$. Symmetrically, B sends $(r', r' \oplus x_B)$ with A's choice y_A, yielding shares of $x_B y_A$. XORing the four shares gives a sharing of xy; flipping one share gives a sharing of $\mathrm{NAND}(x, y) = 1 \oplus xy$. □

Efficiency is dual to FHE: a circuit with G NAND gates and NAND-depth D uses $O(G)$ communication and local work, and $O(D)$ rounds with layerwise parallelism.

A complementary approach uses *garbled circuits*: one party sends a one-time encoding of the circuit and inputs, after which evaluation is noninteractive. Communication scales with circuit size and each execution needs a fresh garbling, but fewer rounds are required.

For large dense linear layers this communication can dominate, which is why 2PC is especially attractive for smaller non-linear steps such as comparisons, ReLU, and top-k.

4 Private vector search

Well, all right, I'll grant you encryption, homomorphic encryption, authentication, obfuscation, secure computation, and even zero-knowledge. But *what has cryptography ever done for us*?

In *private vector search*, the functionality computed is

$$F(\mathbf{q}, \mathbf{M}) = \big(\mathrm{TopK}_k(\mathbf{Mq}), \perp\big).$$

Here $\mathrm{TopK}_k(\mathbf{v})$ returns the k largest coordinates and their indices. The server holds $\mathbf{M}$, the client $\mathbf{q}$, and normalized rows $\mathbf{m}_i$ are scored by $\langle \mathbf{m}_i, \mathbf{q} \rangle = \cos\theta_i$, yielding the k most similar vectors. Specializing to this functionality can greatly reduce the overhead of generic secure computation. In

plaintext, all similarities are computed by the matrix–vector product $\mathbf{Mq}$; on modern ML hardware, its cost is dominated by streaming $\mathbf{M}$ through memory. Once $\mathbf{Mq}$ is available securely, its top k entries can be extracted with $O(N \log k)$ comparisons, essentially linear in N for small k.

The similarity vector itself should not be revealed. Instead, keep $\mathbf{Mq}$ shared between the parties and use 2PC to reveal only its top k entries. Comparing two ℓ-bit scores costs $O(\ell)$ NAND gates, and a selection network uses $O(N \log k)$ comparisons. Thus the large linear scoring stage can be handled separately from the much smaller non-linear top-k postprocessing.

4.1 Private information retrieval

A simpler primitive is *private information retrieval* (PIR). In a *secret-key* setting, the client preprocesses $\mathbf{x} = (x_1, \ldots, x_N)$ into $\widehat{\mathbf{x}}$, uploads it to the server, and keeps secret state sk. To retrieve x_i, it sends a randomized query $\mathbf{q}_i$ to the server and uses sk to decode the server's reply.

Definition 4 (sk-PIR)**.** *An sk-PIR protocol lets the client recover x_i from a server storing $\widehat{\mathbf{x}}$, using $o(N)$ online communication, with $\mathbf{q}_i \approx_c \mathbf{q}_j$ for every i, j.*

Proposition 4 (Secret-key PIR from LPN)**.** *Assuming LPN is hard, for every constant $\varepsilon > 0$ there is sk-PIR with $O(N^\varepsilon)$ online communication.*

Proof. We first explain the algebra over a finite field $\mathbb{F}$. As its secret state, the client chooses a tall systematic matrix and a random mask,

$$\mathbf{D} = \begin{bmatrix} I_N \\ \mathbf{D}' \end{bmatrix}, \qquad \mathbf{r}, \qquad C = \{\mathbf{c} : \mathbf{c}^\top \mathbf{D} = 0\}.$$

Because $\mathbf{D}$ is systematic, the first N coordinates of $\mathbf{Dx}$ are $\mathbf{x}$; the extra rows $\mathbf{D}'$ make C nontrivial. The client sets $sk = (\mathbf{D}, \mathbf{r})$ and uploads

$$\widehat{\mathbf{x}} = \mathbf{Dx} + \mathbf{r}$$

to the server.

To retrieve x_i, the client samples $\mathbf{c} \leftarrow C$ and sends $\mathbf{q}_i = \mathbf{e}_i + \mathbf{c}$, where $\mathbf{e}_i$ selects the ith systematic coordinate. The server returns $a_i = \mathbf{q}_i^\top \widehat{\mathbf{x}}$. Using $\mathbf{r}$, the client computes

$$a_i - \mathbf{q}_i^\top \mathbf{r} = \mathbf{e}_i^\top \mathbf{Dx} + \mathbf{c}^\top \mathbf{Dx} = x_i.$$

This gives correctness, but not privacy: queries $\mathbf{e}_i + \mathbf{c}$ reveal affine translates of the subspace C. The client adds a sparse error $\boldsymbol{\eta}$ and sends $\mathbf{q}_i = \mathbf{e}_i + \mathbf{c} + \boldsymbol{\eta}$. By LPN, these noisy subspace samples are computationally pseudorandom, hiding C and hence i. The noise can violate orthogonality; the construction turns the resulting failures into detectable erasures, corrected by redundancy.

The queries are still long. To reduce communication, reshape the database as a matrix and apply the same idea along one dimension. By linearity, the answer vector is itself an encoding of a smaller database, so the procedure can be folded recursively. Balancing the dimensions and using a constant number of folds gives $O(N^\varepsilon)$ online communication. □

Under stronger assumptions, superconstant folding gives $N^{o(1)}$ communication, with server storage and computation overhead as low as $1.25\times$ the cleartext baseline.

4.2 Encrypted matrix–vector products

An encrypted matrix–vector product (EMVP) is the analogous primitive for a general query vector: after preprocessing $\mathbf{M} \in \mathbb{F}^{N \times d}$, the parties obtain compact shares of $\mathbf{Mq}$ while hiding $\mathbf{M}$ and $\mathbf{q}$.

Proposition 5 (EMVP from LPN)**.** *Assuming LPN is hard, for every constant $\varepsilon > 0$ there is secure EMVP for $\mathbf{M} \in \mathbb{F}^{N \times d}$ with $O(N + d^{1+\varepsilon})$ communication and $O(Nd^{1+\varepsilon})$ server work.*

Proof. The basic construction uses the same orthogonality idea. Let

$$\mathbf{D} = \begin{bmatrix} I_d & \mathbf{D}' \end{bmatrix}, \qquad C = \{\mathbf{c} : \mathbf{Dc} = 0\},$$

and set

$$\widehat{\mathbf{M}} = \mathbf{MD} + \mathbf{R}, \qquad \widehat{\mathbf{q}} = \mathbf{c} + \overline{\mathbf{q}}, \qquad \mathbf{c} \leftarrow C,$$

where $\overline{\mathbf{q}} = (\mathbf{q}^\top, \mathbf{0}^\top)^\top$. Then

$$\widehat{\mathbf{M}}\widehat{\mathbf{q}} = \mathbf{Mq} + \mathbf{R}\widehat{\mathbf{q}}.$$

The server computes the left-hand side. To handle the correction $\mathbf{R}\widehat{\mathbf{q}}$ cheaply, choose $\mathbf{R}$ as a *trapdoored matrix*: it is computationally indistinguishable from random, but the client keeps a trapdoor enabling fast multiplication $\mathbf{v} \mapsto \mathbf{Rv}$. For example, choose a public fast matrix $\mathbf{H}$ from an LPN-hard family; equivalently, for sparse $\mathbf{e}$ and uniform $\mathbf{u}$, $(\mathbf{H}, \mathbf{He}) \approx_c (\mathbf{H}, \mathbf{u})$. Let $\mathbf{E}$ be a secret sparse matrix and set $\mathbf{R} = \mathbf{HE}$. Applying the LPN guarantee to its columns makes $\mathbf{R}$ pseudorandom, while knowing $\mathbf{E}$ lets the client compute $\mathbf{Rv} = \mathbf{H}(\mathbf{Ev})$ quickly. As in PIR, the actual query is noisy so that its distribution hides C under LPN.

Folding along the d query coordinates yields these bounds: the same encrypted query is reused across all N rows, giving $O(d^{1+\varepsilon})$ query communication, $O(N)$ output, and $O(d^{1+\varepsilon})$ server work per row. □

With stronger hidden-subspace assumptions, the same construction can be tuned much closer to the cleartext baselines: communication approaches the $O(N + d)$ input–output size and server work approaches the $O(Nd)$ cost of computing $\mathbf{Mq}$ in the clear.

5 Private neural-network inference

The same linear/non-linear split appears in deep networks. A layer alternates a linear map

$$\mathbf{z}_\ell = \mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell$$

with an activation $\mathbf{x}_\ell = \varphi(\mathbf{z}_\ell)$. One can stay inside FHE by replacing φ with a low-degree polynomial, for example by a Maclaurin expansion. For the sigmoid $\sigma(x) = 1/(1 + e^{-x})$,

$$\sigma(x) \approx p_3(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} \qquad (x \text{ near } 0).$$

This uses only additions and multiplications, trading approximation error for multiplicative depth. Alternatively, evaluate the large linear maps $\mathbf{W}_\ell \mathbf{x}_{\ell-1}$ with EMVP, and use 2PC only for the smaller non-linear steps such as comparisons, ReLU, and taking the maximum over groups of activations.

This is analogous to the principle we applied in secure vector search: use the cheapest cryptographic tool for each part rather than one generic primitive for the whole model.

EMVP also opens the possibility of a hybrid architecture in which the client sits in a trusted environment physically close to the GPU and interacts with the GPU round by round.

6 Historical notes

Goldwasser and Micali introduced probabilistic encryption and the modern computational definition of encryption security in their foundational work [1]. Yao initiated general secure two-party computation in 1982 [2]. Goldreich, Micali, and Wigderson gave a general completeness theorem for secure multiparty computation with an honest majority in 1987 [3]. The OT-based gate-by-gate construction presented here follows Goldreich and Vainish [4]. The complementary garbled-circuit approach is due to Yao [12]. Gentry constructed the first fully homomorphic encryption scheme in 2009 [5]; Brakerski and Vaikuntanathan later gave FHE based on standard LWE [6].

Private information retrieval was introduced by Chor, Goldreich, Kushilevitz, and Sudan [7]. The secret-key PIR construction here is due to Chen, Ishai, Mour, and Rosen [8]; its communication reduction follows the folding paradigm of Ishai and Paskin [13], with superconstant folding requiring stronger hidden-subspace hardness. The EMVP construction is due to Benhamouda et al. [9]. Trapdoored matrices were developed by Vaikuntanathan and Zamir [10] and Braverman and Newman [11]; the fastest known implementation is [9].

References

[1] S. Goldwasser and S. Micali. *Probabilistic Encryption*. Journal of Computer and System Sciences, 28(2), 1984, pp. 270–299.

[2] A. C.-C. Yao. *Protocols for Secure Computations*. FOCS, 1982, pp. 160–164.

[3] O. Goldreich, S. Micali, and A. Wigderson. *How to Play Any Mental Game – A Completeness Theorem for Protocols with Honest Majority*. STOC, 1987, pp. 218–229.

[4] O. Goldreich and R. Vainish. *How to Solve Any Protocol Problem – An Efficiency Improvement*. CRYPTO, 1987, pp. 73–86.

[5] C. Gentry. *Fully Homomorphic Encryption Using Ideal Lattices*. STOC, 2009, pp. 169–178.

[6] Z. Brakerski and V. Vaikuntanathan. *Efficient Fully Homomorphic Encryption from (Standard) LWE*. FOCS, 2011, pp. 97–106.

[7] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. *Private Information Retrieval*. FOCS, 1995, pp. 41–50.

[8] C. Chen, Y. Ishai, T. Mour, and A. Rosen. *Secret-Key PIR from Random Linear Codes*. STOC, 2026, pp. 2019–2029.

[9] F. Benhamouda, C. Chen, S. Halevi, Y. Ishai, H. Krawczyk, T. Mour, T. Rabin, and A. Rosen. *Encrypted Matrix-Vector Products from Secret Dual Codes*. ACM CCS, 2025, pp. 394–408.

[10] V. Vaikuntanathan and O. Zamir. *Improving Algorithmic Efficiency using Cryptography: Trapdoored Matrices and Applications*. SODA, 2026, pp. 2554–2574.

[11] M. Braverman and S. Newman. *Practical Secure Delegated Linear Algebra with Trapdoored Matrices*. TCC, 2025, pp. 97–118.

[12] A. C.-C. Yao. *How to Generate and Exchange Secrets*. FOCS, 1986, pp. 162–167.

[13] Y. Ishai and A. Paskin. *Evaluating Branching Programs on Encrypted Data*. TCC, 2007, pp. 575–594.