In El Gamal encryption, we can calculate an encryption of a product of two messages from their ciphertexts only. Recall that encryptions have the form $Enc(PK, M) = (g^R, PK^R \cdot M)$ for a random R, so given encryptions (h, k) and (h', k') of messages M and M', their pointwise product (hh', kk') gives an encryption of MM'. This property is called *homomorphism*.

Homomorphisms are useful for implementing certain multiparty computations. Suppose a large number of voters want to elect Alice or Bob as their leader. Usually an election committee collects the private votes, tallies them, and publish a count. This amounts to a multiparty computation of the function

$$f(x_1, \ldots, x_n) = x_1 + \cdots + x_n$$

where the inputs represent the voters' choices, say 1 for Alice, -1 for Bob, 0 if neutral.

Physical elections are messy and sometimes violent so there has been some interest in making the process more digital. In principle this can be done securely with the methods from the last few lectures, but you would imagine that any reasonable implementation would be quite inefficient especially when there are many voters.

Here is an alternative. The election committee generates a key-pair for El Gamal encryption and posts the public key. Each voter then posts an encryption $C_i = Enc(PK, g^{x_i})$ of their vote (together with a zero-knowledge proof that x_i takes one of the values -1, 0, or 1). The committee then calculates an encryption of the product $g^{x_1} \cdots g^{x_n} = g^{x_1 + \cdots + x_n}$ and decrypts it. It finds and posts the vote tally by a brute-force discrete log calculation, which is easily feasible if n is in the millions or so.

Besides its practicality, this mechanism has several desirable features. Message indistinguishability of encryptions guarantees that votes are anonymous relative to the other voters. The election committee can convince the voters that the count was carried out properly by providing a zero-knowledge proof of knowledge.

In 1978 Rivest, Adleman, and Dertouzos asked if there are encryptions that support homomorphism of multiplication *and* addition. Their intended application was secure outsourcing of computation. Suppose Bob wants Alice to carry out some long computation for him but he wants to keep his input secret. Bob sends Alice encryptions of her inputs under his public key. Alice then homomorphically evaluates the (arithmetic) circuit representing the computation, eventually obtaining an encryption of the output, which she then sends to Bob. Bob decrypts to obtain his output and Alice hasn't learned anything.

1 Fully homomorphic encryption

An encryption scheme is fully homomorphic if there is a homomorphic evaluation algorithm that, given a circuit f and encryptions of its inputs, outputs an encryption of its outputs. There are two variants of the definition. In the weaker one, the output of the homomorphic evaluator is required to decrypt correctly.

Definition 1. Algorithm Hom is a *weak homomorphic evaluator* for a given encryption scheme if for every circuit f and inputs $x_1, \ldots, x_n$, $Dec(SK, C) = f(x_1, \ldots, x_n)$, where

$$C = Hom(PK, f, Enc(PK, x_1), \ldots, Enc(PK, x_n)).$$

This is sufficient for secure outsourcing of computation. The evaluator Alice gets no information beyond the encryptions of Bob's input, which she can simulate on her own. The outsourcing protocol extends naturally to this two-party computation proposal:

Two-party computation from homomorphic encryption

1. Bob generates SK, PK and sends encryptions $B_1 = Enc(PK, y_1), \ldots, B_n = Enc(PK, y_n)$ of his inputs together with PK to Alice.
2. Alice creates encryptions $A_1 = Enc(PK, x_1), \ldots, A_n = Enc(PK, x_n)$ of her own inputs, computes $C = Hom(PK, f, A_1, \ldots, A_n, B_1, \ldots, B_n)$ and sends C to Bob.
3. Bob outputs $Dec(SK, C)$.

This implementation is still secure for Bob, but it is not clear why it should be secure for Alice. It could be that the ciphertext C contains Alice's input in some part that is ignored by the decryption algorithm. To rule out this possibility, the stronger definition imposes the additional security requirement that the output of the homomorphic evaluator looks like an encryption of $f(x_1, \ldots, x_n)$ that is independent of its inputs.

Definition 2. Hom is a *strong homomorphic evaluator* if the random variables

$(PK, Enc(PK, x_1), \ldots, Enc(PK, x_n), C)$ and $(PK, Enc(PK, x_1), \ldots, Enc(PK, x_n), Enc(PK, y))$

are ε-statistically close, where y and C are the outputs of f and Hom, respectively.

Definition 2 is stronger than Definition 1 because the decryption algorithm itself can be used as a distinguisher. This distinguisher is not efficient since it needs to sample a secret key consistent with the given public key (which is one reason why the indistinguishability requirement is statistical).

The above two-party computation protocol is honest-but-curious secure assuming Hom is strongly secure because Bob can simulate his view from his output. The protocol can be extended to multiple parties. Here is a 3-party computation that is secure with respect to any pair of honest-but-curious parties.

Three-party computation from homomorphic encryption

1. Alice and Bob run a two-party computation to generate a public key PK and private random shares SK_A and SK_B that add up to the secret key SK.
2. Alice and Bob publish PK and send encryptions of their input to Charlie.
3. Charlie encrypts his own inputs, performs the homomorphic evaluation, and publishes the encrypted output C.
4. Alice and Bob run the two-party computation $Dec(SK_A + SK_B, C)$. Bob privately outputs the outcome.

Besides the improvement in security, another advantage over the BGW protocol from last lecture is that the number of messages exchanged is independent of the size of the circuit being evaluated. In summary, fully homomorphic encryption provides simple and efficient solutions to difficult cryptographic problems.

In 2008 Gentry proposed the first candidate homomorphic encryption scheme. This was a significant proof of concept that has since been simplified and improved in several ways. Despite remarkable progress current proposals are still barely practical and rely on non-standard security assumptions. To explain how these work we first need to revisit LWE-based encryption.

2 Learning with errors revisited

Let's start with an LWE-based encryption that is closely related to the key exchange algorithm from Lecture 4. We will be underlining all bounded objects (scalars, vectors, matrices) because boundedness will be important for the discussion. The secret key is a bounded vector $\underline{s}$ and the public key PK is the shortLWE instance $(A, b = \underline{e} + \underline{s}A)$. Then $\underline{s}A - b$ is equal to $\underline{e}$. In matrix form this equation can be written as

$$\underline{sk} \cdot PK = \begin{bmatrix} \underline{s} & -1 \end{bmatrix} \cdot \begin{bmatrix} A \\ \underline{e} + \underline{s}A \end{bmatrix} = -\underline{e},$$

where $\underline{sk}$ is the vector $(\underline{s}, -1)$. To encrypt a message m Bob sends $(A\underline{x} + \underline{e}', b\underline{x} + \underline{e}'' + m)$ to Alice. In matrix form, the encryption is given by

$$Enc(PK, m) = PK \cdot \underline{x} + \begin{bmatrix} \underline{e}' \\ \underline{e}'' \end{bmatrix} + \begin{bmatrix} 0 \\ m \end{bmatrix}.$$

To decrypt $C = Enc(PK, m)$, Alice calculates $\underline{sk} \cdot C$ to obtain

$$\underline{sk} \cdot C = \underline{sk} \cdot PK \cdot \underline{x} + \underline{sk} \cdot \begin{bmatrix} \underline{e}' \\ \underline{e}'' \end{bmatrix} + \underline{sk} \cdot \begin{bmatrix} 0 \\ m \end{bmatrix} = \underline{e} \cdot \underline{x} + \underline{sk} \cdot \begin{bmatrix} \underline{e}' \\ \underline{e}'' \end{bmatrix} - m.$$

If the message is represented by the significant bits of m Alice can recover it because all the "noise" adds up to a bounded scalar.

It turns out that the encryption remains secure even if Bob forgets to add the noise $(\underline{e}', \underline{e}'')$ (assuming A has sufficiently many columns). Under this simplification, the decryption algorithm computes $sk \cdot C = \underline{e} \cdot \underline{x} - m$, so the secret key no longer needs to be bounded for decryption to succeed. This is the original LWE-based encryption scheme for which Oded Regev was awarded the 2018 Gödel prize.

Regev encryption. The secret and public keys are the vector and matrix

$$sk = \begin{bmatrix} s & -1 \end{bmatrix} \in \mathbb{Z}_q^n, \qquad PK = \begin{bmatrix} A \\ \underline{e} + sA \end{bmatrix}$$

where $(A, \underline{e} + sA)$ is a (regular) LWE instance with b-bounded noise. To encrypt, Bob chooses a random vector $\underline{x}$ with $-1, 0, 1$ entries and outputs

$$Enc(PK, m) = PK \cdot \underline{x} + \begin{bmatrix} 0 \\ m \end{bmatrix}$$

where m is encoded in the most significant bits. To decrypt C, Alice calculates $sk \cdot C$, flips its sign, and rounds to the closest message. Decryption is correct because

$$sk \cdot C = sk \cdot PK \cdot \underline{x} + \begin{bmatrix} s & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ m \end{bmatrix} = -\underline{e} \cdot \underline{x} - m$$

and the scalar $\underline{e} \cdot \underline{x}$ is bn-bounded in magnitude.

We don't have all the tools to rigorously prove the security of Regev encryption but it will be instructive to sketch the proof anyway. Message simulatability requires that $(PK, Enc(PK, m))$ can be simulated without knowing m. This random variable has the form $(PK, PK \cdot \underline{x} + c(m))$ where $c(m)$ is the column vector $(0, m)$. By the LWE assumption this is indistinguishable from $(R, R \cdot \underline{x} + c(m))$ for a random matrix R. The missing ingredient is the so-called *leftover hash*

lemma which implies that $(R, R \cdot \underline{x})$ is ε-statistically close to a pair of uniform and independent entries (R, r) (provided R has at least $n \log q + 2\log(1/\varepsilon)$ columns). Then $(R, R \cdot \underline{x} + c(m))$ is also ε-statistically close to (R, r), so encryptions are indistinguishable from uniform random strings.

For the proof of security it is irrelevant how the message m is represented in the column vector $c(m)$. If $c(m)$ was instead chosen to be the column vector $(m, 0)$, encryption and decryption would give

$$Enc(PK, m) = PK \cdot \underline{x} + \begin{bmatrix} m \\ 0 \end{bmatrix} \qquad sk \cdot C = sk \cdot PK \cdot \underline{x} + \begin{bmatrix} s & -1 \end{bmatrix} \cdot \begin{bmatrix} m \\ 0 \end{bmatrix} = -\underline{e} \cdot \underline{x} + m \cdot sk_1,$$

where sk_1 is the first entry of the secret key.

Assuming the message is a single bit $\underline{m} \in \{0, 1\}$ it can still be read off from $sk \cdot C$ with good probability: $sk \cdot C$ is bn-close to zero when $\underline{m} = 0$ and bn-close to sk_1 when $\underline{m} = 1$. Unless sk_1 itself happens to be of magnitude at most $2bn$, $\underline{m}$ can be reliably decrypted. Since sk_1 is random, the probability of a decryption failure is at most $4bn/q$ *over the choice of the secret key.*

The failure probability can be reduced exponentially by repeating the encryption independently n times, using a different position to encode the message in every instantiation. To encrypt a m, Bob sends the n-tuple

$$Enc(PK, \underline{m}) = \big(PK \cdot \underline{x_1} + c_1(\underline{m}),\ PK \cdot \underline{x_2} + c_2(\underline{m}),\ \ldots,\ PK \cdot \underline{x_n} + c_n(\underline{m})\big),$$

where $c_i(\underline{m})$ is the column vector with m in position i and zeroes elsewhere, and $\underline{x_1}, \ldots, \underline{x_n}$ are independent random $\{-1, 0, 1\}$-valued column vectors. This encryption can compactly be represented in the matrix form

$$Enc(PK, \underline{m}) = PK \cdot \underline{X} + \underline{m} \cdot \underline{I} \tag{1}$$

where $\underline{X}$ is a random n-row $\{-1, 0, 1\}$-valued matrix and $\underline{I}$ is the $n \times n$ identity matrix.

To decrypt a bit m, Alice computes the vector-matrix product

$$sk \cdot C = sk \cdot PK \cdot \underline{X} + sk \cdot \underline{m} \cdot \underline{I} = -\underline{e} \cdot \underline{X} + \underline{m} \cdot sk \tag{2}$$

then rounds all the entries to 0 or $q/2$ depending on which number is closer modulo q. If $\underline{m} = 0$ and $bn < q/4$ then all entries will round to zero and decryption is correct. If $\underline{m} = 1$ then the probability that all entries round to zero is at most 2^{-n+1} because the first $n - 1$ entries of the secret key are random.[1]

3 Shallow homomorphic encryption

Let's investigate the homomorphic properties of encryption 1. Since every circuit can be built from NOT and AND gates it is enough to study these two operations. In $\mathbb{Z}_q$ arithmetic, NOT $\underline{m}$ is the function $1 - \underline{m}$ while $\underline{m}$ AND $\underline{m'}$ is multiplication $m \cdot m'$, so we are looking for a way to calculate encryptions of differences and products of messages from their respective encryptions.

In this respect equation (2) has a very useful algebraic property. Messages are "approximate" (modular) eigenvalues of their ciphertext matrices, with the secret key being the corresponding "approximate" eigenvector. If the relations $sk \cdot C = \underline{m} \cdot sk$ and $sk \cdot C' = \underline{m}' \cdot sk$ were *exact* then scalar operations on messages could be emulated by the same matrix operations on their corresponding ciphertexts

$$sk \cdot (C - C') = (\underline{m} - \underline{m}') \cdot sk \qquad \text{and} \qquad sk \cdot CC' = \underline{mm}' \cdot sk,$$

[1] The decryption error can be eliminated at a small price in security.

resulting in a fully homomorphic encryption scheme.

Let us now consider the effect of the noise. Assume that C and C' now satisfy the equations

$$sk \cdot C = \underline{m} \cdot sk + \underline{e} \qquad \text{and} \qquad sk \cdot C' = \underline{m}' \cdot sk + \underline{e}'$$

for some b-bounded e and e'. Then

$$sk \cdot (C - C') = (\underline{m} - \underline{m}') \cdot sk + (\underline{e} - \underline{e}')$$

so $C - C'$ represents the message $\underline{m} - \underline{m}'$, but the best we can say about $\underline{e} - \underline{e}'$ is that it is $2b$-bounded so the noise has grown in magnitude. For ciphertext multiplication, we have

$$\begin{aligned} sk \cdot CC' &= (\underline{m} \cdot sk + \underline{e}) \cdot C' \\ &= \underline{m} \cdot sk \cdot C' + \underline{e} \cdot C' \\ &= \underline{m} \cdot (\underline{m}' \cdot sk + \underline{e}') + \underline{e} \cdot C' \\ &= \underline{mm}' \cdot sk + \underline{m} \cdot \underline{e}' + \underline{e} \cdot C'. \end{aligned} \tag{3}$$

Owing to the likely presence of large entries in the matrix C' this expression is not of the desired form $\underline{mm}' \cdot sk$ plus some bounded vector.

This seemingly intractable difficulty can be resolved by an incredibly naive idea: Before multiplying, replace each entry in the matrix C' by its bit decomposition. For example if $q = 8$ and you see a 3 in C replace it by a 0, a 1, and a 1. Miraculously, all that remains is to tweak the encryption to be syntactically compatible with this change. To describe this modification we need a bit of bit decomposition math.

Bit decomposition The bit decomposition function $\underline{D}(z)$ is the function that maps a number $z \in \mathbb{Z}_q$ to its $\log q$-long bit represnentation. For example, $\underline{D}(3) = (0, 1, 1)$ when $q = 8$. Although the function $\underline{D}$ outputs a sequence of bits, we will think of it as a function from $\mathbb{Z}_q$ to $\mathbb{Z}_q^t$ where $t = \log q$. Then $\underline{D}$ has an inverse $D^\dagger \colon \mathbb{Z}_q^t \to \mathbb{Z}_q$ given by

$$D^\dagger(z_{t-1}, \ldots, z_0) = 2^{t-1} z_{t-1} + \cdots + 2z_1 + z_0. \tag{4}$$

The inverse $D^\dagger$ is a linear function (between $\mathbb{Z}_q$-modules), even though $\underline{D}$ itself is not.

The functions $\underline{D}$ and $D^\dagger$ can be applied pointwise to each element of a given vector or matrix, with the output interpreted as a column vector. Applying $\underline{D}$ to a matrix increases the number of columns by a factor of $t = \log q$. For example, when $q = 8$,

$$\text{if } C = \begin{bmatrix} 3 & 2 \\ 0 & 7 \end{bmatrix} \qquad \text{then} \qquad \underline{D}(C) = \begin{bmatrix} \underline{D}(3) & \underline{D}(2) \\ \underline{D}(0) & \underline{D}(7) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

To recover C back from $\underline{D}(C)$, the entries of each row should be divided into triples and each triple should be combined using formula (4). This amounts to left multiplication by the matrix

$$D^\dagger = \begin{bmatrix} 4 & 2 & 1 & & & \\ & & & 4 & 2 & 1 \end{bmatrix}.$$

You can verify that $D^\dagger \cdot \underline{D}(C)$ recovers C.

In general, if C has n rows, the operation $\underline{D}(C)$ can be inverted by left-mutliplying its output with the tn-by-n matrix $D^\dagger$ that has the vector $(2^{t-1}, \ldots, 2, 1)$ arranged diagonally n times (i.e., the tensor product of this vector and the n-by-n identity matrix).

Gentry-Sahai-Waters (GSW) shallow homomorphic encryption. The keys are the same as in Regev encryption. The encryption procedure is

$$Enc(PK, \underline{m}) = PK \cdot \underline{X} + \underline{m} \cdot D^\dagger. \tag{5}$$

The only difference from (1) is that the identity matrix is replaced by the inverse bit decomposition matrix $D^\dagger$. This does not affect security. Regarding decryptability, because $D^\dagger$ contains the identity matrix, the encryption (1) is embedded in those columns of the GSW encryption that are indexed by multiples of t. Therefore the same decryption procedure works after discarding the other columns. If all the columns are preserved, however, left multiplication by the secret key gives

$$sk \cdot C = sk \cdot (PK \cdot \underline{X} + \underline{m} \cdot D^\dagger) = \underline{e} \cdot \underline{X} + \underline{m} \cdot sk \cdot D^\dagger. \tag{6}$$

As before, homomorphic subtraction of ciphertexts is entrywise matrix subtraction. Given ciphertexts C and C' satisfying

$$sk \cdot C = \underline{m} \cdot sk \cdot D^\dagger + \underline{e} \qquad \text{and} \qquad sk \cdot C' = \underline{m}' \cdot sk \cdot D^\dagger + \underline{e}'$$

for noise vectors $\underline{e}, \underline{e}'$, by linearity $C - C'$ and $\underline{m} - \underline{m}'$ satisfy a relation of the same form. If $\underline{e}$ is b-bounded and $\underline{e}'$ is b'-bounded, the noise vector $\underline{e} - \underline{e}'$ is $(b + b')$-bounded.

Homomorphic multiplication of ciphertexts is now given by the formula $C \cdot \underline{D}(C')$. To analyze it we repeat calculation (3) for GSW ciphertexts:

$$\begin{aligned} sk \cdot C \cdot \underline{D}(C') &= (\underline{m} \cdot sk \cdot D^\dagger + \underline{e}) \cdot \underline{D}(C') \\ &= \underline{m} \cdot sk \cdot D^\dagger \underline{D}(C') + \underline{e} \cdot \underline{D}(C') \\ &= \underline{m} \cdot sk \cdot C' + \underline{e} \cdot \underline{D}(C') \\ &= \underline{m} \cdot (\underline{m}' \cdot sk \cdot D^\dagger + \underline{e}') + \underline{e} \cdot \underline{D}(C') \\ &= \underline{mm}' \cdot sk \cdot D^\dagger + \underline{m} \cdot \underline{e}' + \underline{e} \cdot \underline{D}(C'). \end{aligned}$$

This is precisely of the desired form: If $\underline{e}$ and $\underline{e}'$ are b and b'-bounded, respectively, then $\underline{m} \cdot \underline{e}' + \underline{e} \cdot \underline{D}(C')$ is $(nb + b')$-bounded.

To summarize, homomorphic subtraction and multiplication preserve the form of ciphertexts, but the noise magnitude grows with each operation. If a circuit f is evaluated homomorphically gate-by-gate, the ratio between the noise at the output and the noise at the input grows by a factor that is exponential in its depth.[2] If the depth of f is smaller than the logarithm of the modulus q then the noise never grows past the decryption threshold and the output of the homomorphic evaluator decrypts correctly.[3]

4 Denoising

The final component for weak fully homomorphic encryption is a mechanism for denoising ciphertexts. This is a procedure Den that, given a ciphertext C (and the public key), outputs another

[2] The depth of the circuit is the maximum path length in the underlying graph.

[3] More precisely, the depth ought to be at most $\log_n(q/4bn)$ where b is the encryption noise.

ciphertext C' that encrypts the same message but with less noise: If $sk \cdot C = \underline{m} \cdot sk \cdot D^\dagger + \underline{e}$ for some b^+-bounded $\underline{e}$ then $sk \cdot C' = \underline{m} \cdot sk \cdot D^\dagger + \underline{e}'$ for some b^--bounded $\underline{e}'$ with $b^- < b^+$.

So far all the LWE transformations increase the amount of noise, so it is not clear that denoising is at all plausible. Where should we even begin to look? The answer is again incredibly naive:

> Homomorphically evaluate the decryption circuit *as a function of the secret key.*

If you prefer equations to mantras, that translates to

$$Den(PK, C) = Hom(PK, Dec_C, Enc(PK, sk)) \tag{7}$$

where Dec_C is the decryption circuit with the input ciphertext C hardwired into it.

Decryption functionality says that $Dec_C(sk)$ outputs the message m. Homomorphic evaluation functionality says that a homomorphic evaluation of Dec_C given an encryption of sk outputs a new encryption C' of m.

What about noise? As long as $b^+ \leq q/4n$ decryption is guaranteed to succeed. On the other hand, the noise of the homomorphic evaluator's output only depends on the noise of its input and the depth of the decryption circuit. Since the input is an actual encryption its noise is completely determined by the LWE parameters. In the case of GSW encryption, (5) and (6) reveal that the input noise is at most n times the LWE noise bound. Using a suitably shallow implementation of GSW decryption, its contribution can be bounded by some polynomial in n. In conclusion, the amount b^- of noise in C' can be bounded by some fixed polynomial in n that is independent of q.

To implement weak fully homomorphic evaluation, start with shallow evaluation until the upper noise threshold b^+ is reached, denoise down to b^-, and keep going until the output gate is reached.

There is one more issue to be discussed. To implement (7), the denoising circuit needs to know the encryption of the secret key. The only place it could possibly obtain this information from is the public key itself. To support full homomorphic evaluation, the public key of GSW encryption ought to be augmented with an encryption of the (bits of the) secret key. Encryption schemes that are immune against such disclosures are said to satisfy circular security.

Definition 3. A public-key encryption scheme is *circular-secure* if for every message M, $Enc(PK, M)$ is simulatable given PK *and* $Enc(PK, sk)$ (with the usual parameters).

Circular security is not a direct consequence of (multiple) message simulatability. In the definition from Lecture 4 the challenge message does not depend on the secret key. It is not known if El Gamal encryption is circular-secure under the DDH assumption, or if the Regev and GSW encryptions are circular-secure under the LWE assumption, but it appears plausible that they might be.[4]

5 Homomorphic encryption and obfuscation

On several occasions we used virtual black-box (VBB) obfuscation to argue the plausibility of cryptographic constructions like public-key encryption and signatures based on the Fiat-Shamir heuristic. We are now in a position to sketch the proof that general-purpose VBB obfuscation does not exist.

VBB security demands that for every circuit C, the output of any learner that is given an obfuscation $Obf(C)$ is indistinguishable from the output of a simulator that had oracle access to C. For

[4] In contrast, they are secure when the encrypted message depends on the *public* key.

starters we will argue that VBB obfuscation for a pair of circuits is impossible: Namely, there exist circuits P and T and a learner that is given their obfuscations whose output cannot be simulated by oracle access to P and T.

The circuit names P and T stand for point and test, respectively. Their descriptions involve three secret keys in, out, and $feature$ that are chosen uniformly at random from $\{0,1\}^k$. The point circuit maps k-bit inputs to k-bit outputs but evaluates to zero everywhere except at in:

$$P(x) = \begin{cases} out, & \text{if } x = in \\ 0, & \text{if not.} \end{cases}$$

The test circuit takes the description of a circuit C as its input and outputs $feature$ only if C maps in to out:

$$T(C) = \begin{cases} feature, & \text{if } C(in) = out \\ 0, & \text{if not.} \end{cases}$$

Given the circuits P and T, $feature$ can be recovered by running T on P. The same is true if $Obf(T)$ is evaluated on $Obf(P)$, so there is a linear-size learner that outputs $feature$. However, any VBB simulator is very unlikely to output $feature$: To do this, the simulator must either happen to query P on in to get out, or if not query T on a circuit that maps in to out, or if not guess $feature$ at random. Each of these events has probability at most $s \cdot 2^{-k}$ for a simulator of size s.

How about obfuscating a single circuit? To rule this out, it is natural to try and combine P and T into a single hard circuit H. This is usually done by furnishing H with an extra bit of input that tells it whether to operate in P-mode or in T-mode:

$$H(mode, z) = \begin{cases} P(z), & \text{if } mode = \mathtt{P} \\ T(z), & \text{if } mode = \mathtt{T}. \end{cases}$$

To mimic the proof we just gave, we first need to show how to extract $feature$ from an obfuscation $O = Obf(H)$. To do this T should be evaluated on P, which amounts to first printing the circuit $O_{\mathtt{P}}(x) = O(\mathtt{P}, x)$ and then computing the output $O(\mathtt{T}, O_{\mathtt{P}})$. This involves running the circuit O on itself as an input. Unless the size of O is smaller than the length of its input (which means O should be very primitive), O cannot be evaluated on itself.

This is where fully homomorphic encryption comes into play. It allows for the evaluation of the test circuit to be outsourced to the learner. Consider instead the circuit H that outputs

$$H(mode, z) = \begin{cases} PK, Enc(PK, in), & \text{if } mode = \mathtt{I}, \\ out, & \text{if } mode = \mathtt{P} \text{ and } z = in, \\ feature, & \text{if } mode = \mathtt{T} \text{ and } Dec(sk, z) = out, \end{cases}$$

and outputs zero otherwise.

Given the obfuscated circuit $O = Obf(H)$ the learner can now extract $feature$ like this: First, it runs $O(\mathtt{I})$ to obtain PK and the encryption of in. It then evaluates the circuit $O_{\mathtt{P}}$ homomorphically on the encryption of in to obtain an encryption C of out. Finally it runs $O(\mathtt{T}, C)$ to output $feature$.

From the simulator's perspective, oracle access to H is (s, ε)-indistinguishable from oracle access to a circuit that is the same except that it outputs a simulated encryption in mode $\mathtt{I}$. Then queries in mode $\mathtt{I}$ give no information about P or T so H outputs $feature$ with probability at most $s \cdot 2^{-k} + \varepsilon$.

The conclusion is that if fully homomorphic encryption is possible, then VBB obfuscation isn't. It turns out that fully homomorphic encryption can be constructed from VBB obfuscation (without extra assumptions like circular security of LWE-based encryptions), so VBB obfuscation bites its own tail: If it exists, then it doesn't exist.