Hopes, fears, and software obfuscation

Boaz Barak[*]

February 25, 2016

Abstract

I survey some of the recent progress on software obfuscation spurred by the exciting paper of Garg, Gentry, Halevi, Raykova, Sahai and Waters [GGH$^+$13]. This is a preprint version of a review article [Bar16] appearing in the Communications of the ACM. That article was written for a general computing audience, and is also not up to date on the latest research in this fast-moving field since it was mostly written in 2014. Nevertheless, I thought it might still be of interest to cryptographers.

Computer programs are arguably the most complex objects ever constructed by humans. Even understanding a 10-line program such as the one depicted in Figure 1 can be extremely difficult. The complexity of programs has been the bane (as well as the boon) of the software industry, and taming it has been the objective of many efforts in industry and academia. Given this, it is not surprising that both theoreticians and practitioners have been trying to "harness this complexity for good" and use it to protect sensitive information and computation. In its most general form this is known as *software obfuscation*, and it is the topic of this survey.

In a certain sense, any cryptographic tool such as encryption or authentication can be thought of harnessing complexity for security, but with software obfuscation people have been aiming for something far more ambitious: a way to transform *arbitrary* programs into an "inscrutable" or *obfuscated* form. By this we don't mean that reverse engineering the program should be cumbersome but rather that it should be *infeasible*, in the same way that recovering the plaintext of a secure encryption cannot be performed using any reasonable amount of resources.

[*]Harvard John A. Paulson School of Engineering and Applied Sciences, `b@boazbarak.org`. This article was written while the author was at Microsoft Research

```
def isprime(p):
    return all(p % i  for i in range(2,p-1))

def Goldbach(n):
    return any( (isprime(p) and isprime(n-p))
           for p in range(2,n-1))

n = 4
while True:
    if not Goldbach(n): break
    n+= 2
print "Hello world!"
```

Figure 1: *The following Python program prints* `"Hello world!"` *if and only if Goldbach's conjecture is false*

Obfuscation, if possible, could be the cryptographer's "master tool"—as we'll see it can yield essentially any crypto application one can think of. Therefore it is not surprising that both practitioners and theoreticians have been trying to achieve it for a great while. However, over the years many practical attempts at obfuscation (such as the DVD Content Scrambling System) had been broken (e.g., see [JBF03, Gre14]). Indeed in 2001, in work with Impagliazzo, Goldreich, Rudich, Sahai, Vadhan, and Yang [BGI+01], we proved that achieving such a secure transformation is *impossible.* So, why isn't this the shortest survey in CACM history?

The key issue is what does it mean to be "secure". In our 2001 work, we proved impossibility of a notion of security for such "obfuscating compilers", which we termed as *virtual black box security* and will describe more formally in Section 1 below. While virtual black-box is arguably the most natural notion of security for obfuscators, it is not the only one. Indeed in the same work [BGI+01] we suggested a weaker notion called *indistinguishability obfuscation* or IO for short. We had no idea if IO can be achieved, nor if it is useful for many of the intended applications. But in 2013 Garg, Gentry, Halevi, Raykova, Sahai and Waters [GGH+13] used some recent cryptographic advances to give a *candidate construction* of obfuscators satisfying the IO definition. Moreover, their work, and many followup works, have shown that this weaker notion is actually extremely useful, and can recover many (though not all) the applications of virtual black-box obfuscators. These applications include some longstanding cryptographic goals that before Garg et al's work seemed far out of reach, and so the cryptographic community is justifiably excited about these new developments, with many papers and several publicly funded projects devoted to exploring obfuscation and its applications.

What is an indistinguishability obfuscator? how is it useful? and what do I mean by a "candidate construction"? Read the rest of this survey to find out.

1 Obfuscating compilers and their potential applications

Obfuscation research is in many ways in an "embryonic stage", where, as we discuss in Section 2 below, so far we only have theoretical "proofs of concept" that are extremely far from practical efficiency. Even with the breakneck pace of research on this topic, it may take years, if not decades, until such obfuscators can be deployed at scale, and as we'll see, beyond the daunting practical issues there are some fundamental theoretical challenges we'll need to address as well. Thus, while eventually one might hope to obfuscate large multi-part programs, in this survey we focus on the task of obfuscating a single function, mapping an input to an output without any side effects (though there is recent research on obfuscating more general computational models). Similarly, given that the overhead in translating a program from one (Turing complete) programming language to another pales in comparison to the current inefficiencies, for the purposes of this survey we can imagine that all programs are represented in some simple canonical way.

An *obfuscating compiler* is an algorithm that takes as input a program P and produces a functionally-equivalent program P'. So far, this does not rule out the compiler that simply outputs its input unchanged, but we'll want the program P' to be "inscrutable" or "obfuscated". Defining this requirement formally takes some care, and as we will see, there is more than one way to do so. Our guiding principle is that the output program P' will not reveal any more information about the input P than is necessarily revealed by the fact that the two programs are functionally equivalent. To take two extreme examples, if you wrote a program P that outputs your credit card number when given the number 0 as input, then no matter how obfuscated P' is, it will still reveal the same information. In contrast, if the program P contained the credit number as a comment, with no effect on its functionality, then the obfuscated version P' should not reveal it. Of course, we want an obfuscator compiler to do much more than strip all comments. Specifically, we say that a compiler transforming P to P' is *virtual black-box secure* if for any attacker A that learns some piece of information x from P', x could have been learned by simply treating P' (or P) as a *black box* and querying it on various inputs and observing the outputs. More formally, the definition requires that for every (polynomial-time) algorithm A, there is another polynomial time algorithm S (known as a *simulator* in crypto parlance) such that the random variables $A(P')$ and S^P are computationally indistinguishable, where $A(P')$ denotes the output of A given the code of the obfuscated program P', while S^P denotes the output of S given black-box (i.e. input/output) access to P (or to the functionally equivalent P').

Let us imagine that had a practically efficient obfuscating compiler that satisfied this notion of virtual black-box security. What would we use it for? The most obvious application is software protection— publishing an obfuscated program P' would be the equivalent of providing a physical "black box" that can be executed but not opened up and understood. But there are many other applications. For example, we could use an obfuscator to design a "selective decryption scheme"[1]— a program that would contain inside it a decryption key d but would only decrypt messages satisfying very particular criteria. For example, suppose that all my email was encrypted, and I had an urgent project that needed attention while I was on vacation. I could write a program P, such as the one of Figure 2, that given an encrypted email email as input, uses my secret decryption key to decrypt it, checks if it is related to this project and if so outputs the plaintext message. Then, I could give my colleague an obfuscated version P' of P, without fearing that she could reverse engineer the program, learn my secret key and manage to decrypt my other emails as well.

There are many more applications for obfuscation. The example of functional encryption could be vastly generalized. In fact, *almost any cryptographic primitive you can think of* can be fairly directly derived from obfuscation, starting from basic primitives such as public key encryption and digital signatures to fancier notions such as multiparty secure computation, fully homomorphic encryption, zero knowledge proofs, and their many variants. There are also applications to obfuscation that a priori seem to have nothing to do with cryptography; e.g., one can use it to design autonomous agents that would participate on your behalf in digital transactions such as electronic auctions, or to publish patches for software vulnerabilities without worrying that attackers could learn the vulnerabilities by reverse-engineering

[1]The technical name for this notion is *functional encryption* [BSW12].

```
def DecryptEmail(EncryptedMsg):
    SecretKey = "58ff29d6ad1c33a00d0574fe67e53998"
    m = Decrypt(EncryptedMsg,SecretKey)
    if m.find("Foosball table")>=0: return m
    return "Sorry Yael, this email is private"
```

Figure 2: *A "selective decryption" program. Black-box access to this program enables a user to decrypt only messages that match some particular pattern, and hence obfuscating such a program can be used to obtain a* functional encryption scheme*.*

the patch.

So, virtual black-box obfuscation is wonderful, but does it exist? This is what we set to find out in 2001, and as already mentioned, our answer was negative. Specifically, we showed the existence of *inherently unobfuscatable functions*— this is a program P whose source code can be recovered from *any* functionally equivalent program P' though curiously it *cannot* be efficiently recovered using only black-box access to P.

In the intervening years, cryptography has seen many advances, in particular achieving constructions of some of the cryptographic primitives that were envisioned as potential applications of obfuscation, most notably *fully homomorphic encryption* [Gen09] (see also Section 3 and Figure 4).In particular, in 2012 Garg, Gentry and Halevi [GGH12] put forward a candidate construction for an object they called "cryptographic multilinear maps", and which in this survey I'll somewhat loosely refer to as a "homomorphic quasi encryption" scheme. Using this object, Garg et al [GGH$^+$13] showed a candidate construction of a general-purpose *indistinguishablity obfuscators.*[2]

2 Indistinguishability obfuscators

An *indistinguishability obfuscator (IO)* hones in on one property of virtual black box obfuscators. Suppose that P and Q are two functionally-equivalent programs. It is not hard to verify that virtual black-box security implies that an attacker should not be able to tell apart the obfuscation P'

[2]Even prior to the works [GGH12, GGH$^+$13] there were papers achieving virtual black-box obfuscation for very restricted families of functions. In particular, independently of [GGH$^+$13], Brakserski and Rothblum [BR13] used [GGH12]'s construction to obtain virtual black-box obfuscation for functions that can be represented as *conjunctions* of input variables or their negations.

of P from the obfuscation Q' of Q. Indistinguishability obfuscation requires only this property to hold.

Indistinguishability obfuscators were first defined in our original paper [BGI$^+$01], where we noted that this notion is weak enough to avoid our impossibility result, but we did not know whether or not it can be achieved. Indeed, a priori, one might think that indistinguishable obfuscators capture the "worst of both worlds". On one hand, while the relaxation to IO security does allow to avoid the impossibility result, such obfuscators still seem incredibly hard to construct. For example, assuming Goldbach's Conjecture is correct, the IO property implies that the obfuscation of the `Goldbach(n)` subroutine of the program in Figure 1 should be indistinguishable from the obfuscation of the function that outputs `True` on every even $n > 2$; designing a compiler that would guarantee this seems highly non-trivial. On the other hand, it is not immediately clear that IO is useful for concrete applications. For example, if we consider the "selective decryption" example mentioned above, it is unclear that the IO guarantee means that obfuscating the program P that selectively decrypts particular messages would protect my secret key. After all, to show that it does, it seems we would need to show that there is a functionally-equivalent program P' that does not leak the key (and hence by the IO property, since P and P' must have indistinguishable obfuscations, the obfuscation of P would protect the key as well). But if we knew of such a P', why didn't we use it in the first place?

It turns out that both these intuitions are (probably) wrong, and that in some sense IO may capture the "best of both worlds". First, as I mentioned, despite the fact that it seems so elusive, Garg et al [GGH$^+$13] did manage to give a candidate construction of indistinguishability obfuscators. Second, [GGH$^+$13] managed to show that IO is also *useful* by deriving functional encryption from it, albeit in a less direct manner. This pattern has repeated itself several times since, with paper after paper showing that many (though not all) of the desirable applications of virtual black-box obfuscation can be obtained (using more work) via IO. Thus indistinguishability obfsucation is emerging as a kind of "master tool" or "hub" of cryptography, from which a great many of our other tools can be derived (see Figure 3).

So now all that is left is to find out how do we construct this wonderful object, and what is this caveat of "candidate construction" that I keep mentioning?

I will start with describing the construction and turn later to discussing the caveat. Unfortunately, the construction is rather complex. This is both in the colloquial sense of being complicated to describe (not to mention im-

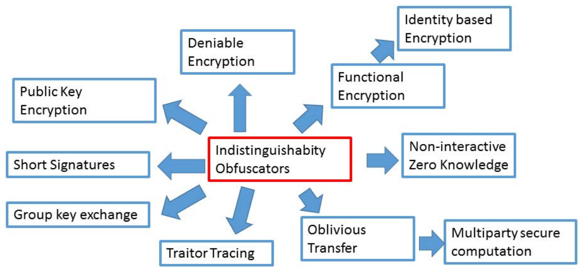


Figure 3: In the two years since the candidate construction, Indistinguishability Obfuscation is already emerging as a "hub" for cryptography, implying (when combined with one way functions) a great many other cryptography primitives.

plement) and in the computational complexity sense of requiring very large (though still polynomial) space and time resources. Indeed, this complexity is the main reason these constructions are still at the moment theoretical "proof of concepts", as opposed to practical compilers. The only implementation of obfuscators I know of at the time of this writing was by Apon et al [AHKM14], and their obfuscation blows up a circuit of 16 OR gates to $31GB$. (The main source of inefficiency arises from the constructions of "homomorphic quasi-encryption schemes" described below.) That said, making these schemes more efficient is the object of an intensive research effort, and I am sure that we will see many improvements in the coming years. It is a testament to the excitement of this field that in the short time after the first candidate construction of IO, there are already far more works than I can mention that use IO for exciting applications, study its security or efficiency, consider different notions of obfuscation, and more.

While I will not be able to describe the actual construction, I do hope to give some sense into the components that go into it, and the rather subtle questions that arise in exploring its security.

3 Fully homomorphic encryption and "quasi encryption"

In 2009, Craig Gentry rocked the world of cryptography by presenting a construction for *fully homomorphic encryption scheme*. What is this object? Recall that a traditional encryption scheme is composed of two func-

tions: the encryption operation Enc— mapping the secret plaintexts into the "scrambled" cyphertexts— and the decryption operation Dec that performs the inverse of Enc (and requires the secret decryption key to compute). A fully homomorphic encryption supports two additional operations $\oplus, \otimes$ which correspond to "multiplying" and "adding" ciphertexts. Specifically they satisfy the equations for every $a, b \in \{0,1\}$: Enc(a) $\oplus \mathsf{Enc}(b) = \mathsf{Enc}(a + b \pmod 2)$)
$\mathsf{Enc}(a) \otimes \mathsf{Enc}(b) = \mathsf{Enc}(ab \pmod 2)$)

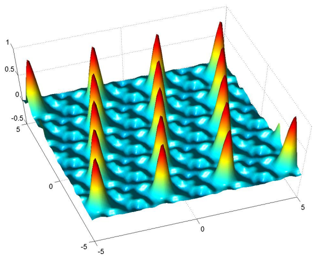
Since NOT a = 1 + a (mod 2)
$aANDb = ab \pmod 2$
$a\ OR\ b = a + b + ab \pmod 2$ these two operations allow us to compute from encryptions $\mathsf{Enc}(a_1), \ldots, \mathsf{Enc}(a_n)$ the value $\mathsf{Enc}(P(a_1, \ldots, a_n))$ given any program P that maps $\{0,1\}^n$ to $\{0,1\}$. Note that crucially, the $\otimes$ and $\oplus$ operations do *not* require knowledge of the secret key to be computed (indeed, otherwise they would be trivial to implement by writing c $\oplus c' =$ $\mathsf{Enc}(\mathsf{Dec}(c) + \mathsf{Dec}(c') \pmod 2)$)
$c \otimes c' = \mathsf{Enc}(\mathsf{Dec}(c)\mathsf{Dec}(c') \pmod 2)$)

Fully homomorphic encryption was first envisioned in 1978 by Rivest, Adleman, and Dertouzos [RAD78], but they gave no constructions and for many years it was not at all clear whether the existence of such operations is compatible with security until Gentry [Gen09] came up with the first plausibly-secure construction. [RAD78] were motivated by client-server applications (now known as "cloud computing"). Indeed one can see that such an encryption scheme could be very useful in this setting, where for example a client could send to the server an encryption $\mathsf{Enc}(a) = \mathsf{Enc}(a_1) \cdots \mathsf{Enc}(a_n)$ of its private data a, so that the server could use the $\oplus$ and $\otimes$ operations to compute some complicated program P on this encryption and return $\mathsf{Enc}(P(a))$ to the client, without ever learning anything about a.

The astute reader might notice that fully homomorphic encryption is an immediate consequence of (virtual black-box) obfuscation combined with any plain-old encryption. Indeed, if secure obfuscation existed then we could implement $\oplus$ and $\otimes$ by obfuscating their trivial programs (3) and (3). One might hope that would also work in the other direction— perhaps we could implement obfuscation using a fully homomorphic encryption. Indeed let F be the "program interpreter" function that takes as input a descrptyion of the program P and a string a and maps them to the output $P(a)$. Perhaps we could obfuscate the program P by publishing an encryption $P' = \mathsf{Enc}(P)$ of the description of P via a fully homomorphic encryption. The hope would be that we could use P' to evaluate P on input a by encrypting a and then invoking F on the encrypted values $\mathsf{Enc}(P)$ and $\mathsf{Enc}(a)$ using the

While the integer factoring problem is perhaps the most well known mathematical basis for cryptosystems, many recent constructions, including those used by fully homomorphic encryption and obfuscation, use computational problems related to *integer lattices*.

The fundamental observation behind these problems is that classical linear algebraic algorithms such as Gaussian elimination are incredibly *brittle* in the sense that they cannot handle even slight amounts of noise in their data. One concrete instantiation of this observation is Regev's *Learning With Errors* (LWE) conjecture [Reg05] that there is no efficient algorithm that can recover a secret random vector $x \in \{0, \dots, p-1\}^n$ given noisy linear equations on x (i.e., a random matrix A and the vector $y = Ax + e \pmod p$ where e is a random error vector of small magnitude). This has been shown to be essentially equivalent to the question of trying to "error correct" a vector in $\mathbb{R}^n$ that sampled from a distribution that is very close to, but not exactly contained in, a discrete subspace (i.e., a *Lattice*) of $\mathbb{R}^n$.



The LWE problem turns out to be an even more versatile basis for cryptography than discrete log and integer factoring and it has been used as a basis for a great many cryptographic schemes. It also has the advantage that, unlike factoring and discrete log, it is not known to be breakable even by *quantum* computers.

We now give a very rough sketch of how LWE can be used to obtain a fully homomorphic encryption scheme, following the paper [GSW13]. See Gentry's excellent survey [Gen14] for an accessible full description of this scheme. The basic starting point of [GSW13]'s scheme is the following candidate encryption: the secret key is some vector $s \in \{0, \dots, p-1\}^n$, and to encrypt the message $\lambda \in \{0, \dots, p-1\}$ we generate a random matrix A such that $As = \lambda s \pmod p$. Note that this scheme is obviously homomorphic— if $As = \lambda s \pmod p$ and $A's = \lambda' s \pmod p$ then $(A + A')s \pmod p = (\lambda + \lambda')s \pmod p$ and $(AA')s = \lambda\lambda' s \pmod p$. Unfortunately, it is also obviously insecure— using Gaussian elimination we can recover s from sufficiently many encryptions of zero. [GSW13] fix this problem by adding *noise* to these encryptions, hence fooling the Gaussian elimination algorithm. Managing the noise so that it doesn't blow up too much in the homomorphic operations requires delicate care and additional ideas, and this is the reason why Gentry called his survey "computing on the edge of Chaos".

Figure 4: Lattice based cryptography and fully homomorphic encryption

homomorphic operations. However, a moment's thought shows that if we do that, we would not get the value $P(a)$ but rather the *encryption* of this value. Thus P' is not really a functionally equivalent form of P, as (unless one knows the decryption key) access to P' does not allow to compute P on chosen inputs. Indeed, while fully homomorphic encryption does play a part in the known constructions of obfuscation, they involve many other components as well.

In some sense, the problem with using a fully homomorphic encryption scheme is that it is "too secure". While we can perform various operations on ciphertexts, without knowledge of the secret key we do not get any information at all about the plaintexts, while obfuscation is all about the "controlled release" of particular information on the secret code of the program P. Therefore the object we need to construct is what I call a "fully homomorphic quasi-encryption" which is a riff on an encryption scheme that is in some sense less secure but more versatile than a standard fully homomorphic encryption.[3]

3.1 Homomorphic quasi-encryption

A "fully homomorphic quasi-encryption scheme" has the same Enc, Dec, $\otimes$ and $\oplus$ operations as a standard fully homomorphic encryption scheme, but also an additional operation $\stackrel{?}{=}$ which satisfies that $\mathsf{Enc}(a) \stackrel{?}{=} \mathsf{Enc}(b)$ is `true` if $a = b$ and equals `false` otherwise. Moreover, instead $\{0, 1\}$, the plaintexts will be in $\{0, 1, \ldots, p - 1\}$ for some very large prime p (of a few thousand digits or so), and the $\oplus$ and $\otimes$ operations are done modulo p instead of modulo 2. Note that a quasi-encryption scheme is less secure than standard encryption, in the sense that the $\stackrel{?}{=}$ operation allows an attacker to perform tasks, such as discovering that two ciphertexts correspond to the same plaintext, that are infeasible in a secure standard encryption. Indeed, the notion of security for a quasi-encryption scheme is rather subtle and is very much still work-in-progress.[4] A necessary condition is that one should not be able to recover a from $\mathsf{Enc}(a)$ but this is in no way sufficient. For now let us say that the quasi encryption scheme is secure if an attacker

[3]This is a non-standard name used for this exposition; the technical name is a *cryptographic multilinear map* or a *graded encoding scheme*.

[4]In fact, the same paper [GGH12] proposing the first candidate construction for such a quasi-encryption scheme also gave an attack showing that their scheme does *not* satisfy some natural security definitions, and followup works extended this attack to other settings. Finding a candidate construction meeting a clean security definition that suffices for indistinguishability obfuscation is an important open problem.

cannot learn anything about the plaintexts beyond what could be learned by combining the $\otimes, \oplus$ and $\stackrel{?}{=}$ operations.[5]

One example of a "partially homomorphic quasi-encryption" scheme is *modular exponentiation.* That is, given some numbers g, q such that $g^p = 1 \pmod q$, we can obtain a quasi-encryption scheme supporting only $\oplus$ (and not $\otimes$) by defining $\mathsf{Enc}(a) = g^a \pmod q$ for $a \in \{0, \ldots, p-1\}$ (with Dec its inverse— i.e. the discrete log modulo q).[6] We define $c \oplus c' = cc' \pmod q$ which indeed satisfies that $g^a \oplus g^{a'} = g^{a+a'}$, and define $\stackrel{?}{=}$ to simply check if the two ciphertexts are equal. Modular exponentiation has been the workhorse of cryptography since Diffie and Hellman (following a suggestion of John Gill) used it as a basis for their famous key exchange protocol [DH76]. In 2000 Joux [Jou00] suggested (using different language) to use exponentiation over elliptic curve groups which support the so called "Weil and Tate pairings" to extend this quasi-encryption scheme to support a single multiplication. Surprisingly even a single multiplication turns out to be extremely useful and a whole sub-area of cryptography, known as "pairing based cryptography", is devoted to using these partially homomorphic quasi-encryptions for a great many applications. But the grand challenge of this area has been to obtain *fully homomorphic* quasi-encryption [BS02] (or in their language a *multi-linear* map, as opposed to the *bi-linear* pairing). An exciting approach toward this grand challenge was given by the work of Garg, Gentry and Halevi [GGH12]. On a very high level, they showed how one can modify a fully homomorphic encryption scheme to support the $\stackrel{?}{=}$ operation by publishing some partial information on the secret decryption key, that at least as far as we know, only allows to check for plaintext-equality of ciphertexts without revealing any additional information. The main challenge remaining is that the security of their scheme has yet to be proven (and in fact we have yet to even find the right *definitions* for security). I discuss this issue more below in Section 4. But even with this caveat their work is still a wonderful achievement, and provides cryptography with a candidate construction for one of the most versatile tools with which one can achieve a great many cryptographic objectives.

[5]As an aside, a similar notion to quasi encryption, without homomorphism, is known as *deterministic encryption* and is used for tasks such as performing SQL queries on encrypted data bases (e.g., see [PRZB12]).

[6]The Dec operation is not efficiently computable, but this turns out not to be crucial for many of the applications.

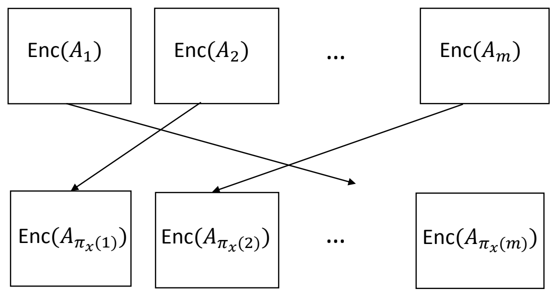


Figure 5: We use Barrington's Theorem to encode a program F computable by a logarithmic depth circuit into a sequence of m matrices $A_1, \ldots, A_m$ and publish the quasi-encryptions of these matrices' entries. Every input x corresponds to a permutation π_x such that if we multiply the matrices in this order, the top left element in the resulting matrix will equal $F(x)$.

3.2 From quasi-encryption to obfuscation

As mentioned, the construction of obfuscation from a fully homomorphic quasi-encryption is rather complicated, but I will attempt to outline some of the ideas behind it. I will not even try to argue about the *security* of the obfuscation construction, but rather simply give some hints of how one might use the quasi-encryption scheme to represent a program P in a form that at least intuitively seems very hard to understand. At a high level the obfuscation of a program P consists simply of the quasi-encryptions (which we'll call *encodings*) of N numbers $a_1, \ldots, a_N$. To make this a valid representation, we need to supply a way to compute $P(x)$ from these encodings for every input x. The idea is that every input x would correspond to some formula f_x involving additions and multiplications modulo p such that $P(x) = 0$ if and only if $f_x(a_1, \ldots, a_m) = 0 \pmod{p}$. Since we can test the latter condition using the $\oplus$, $\otimes$ and $\overset{?}{=}$ operations, we can find out if $P(x) = 0$ or $P(x) = 1$. This results in an obfuscation of programs with one bit of output, but can be generalized to handle programs with larger outputs.

How do we construct this magical mapping of inputs to formulas? We cannot present it fully here, but can describe some of the tools it uses. One component is the naive approach described above of constructing an obfuscation scheme from a fully homomorphic encryption. As we noted, this approach does not work because it only allows to compute the output of the program in encrypted form, but it does essentially reduce the task of obfuscating an *arbitrary* function to the task of obfuscating the *decryption function* of the concrete encryption scheme. The crucial property for us is that this decryption function can be computed via a *logarithmic depth circuit*. This allows us to use some of the insights that have been obtained in the study of logarithmic depth circuits (which had been developed towards obtaining circuit lower bounds, without envisioning any cryptographic or other practical applications whatsoever). In particular, Barrington [Bar86] proved in 1986 the following beautiful but seemingly completely abstract result (see also Figure 5):

Theorem 1 *If $F : \{0,1\}^n \to \{0,1\}$ is a function computable by a log-depth circuit, then there exists a sequence of $m = \mathrm{poly}(n)$ 5×5 matrices $A_1, \ldots, A_{m'}$ with entries in $\{0,1\}$ and a mapping $x \mapsto \pi_x$ from $\{0,1\}^n$ into the set of permutations of $[m']$ such that for every $x \in \{0,1\}^n$*

$$F(x) = \left(\prod_{i=1}^{m'} A_{\pi_x(i)} \right)_{1,1} . \qquad (1)$$

(That is, $F(x)$ is equal to the top left element of the product of matrices according to the order π_x.)

This already suggests the following method for obfuscation: if we want to obfuscate the decryption function F, then we construct the corresponding matrices $A_1, \ldots, A_m$, encode all $N = 25m$ of their entries (which we will call $a_1, \ldots, a_N$), and then define for every x the formula f_x to be the right-hand side of (1). This is a valid representation of the program P, since by using the homomorphic properties of the quasi-encryption we can compute from the N encodings of numbers the value of $F(x)$ (and hence, by combining this with our previous idea, also the value of $P(x)$) for every input x. However, it is not at all clear that this representation doesn't leak additional information about the function. For example, how can we be sure that we cannot recover the secret decryption key by multiplying the matrices in some different order?

Indeed, the actual obfuscation scheme of [GGH$^+$13] is more complicated and uses additional randomization tricks (as well as a more refined variant of quasi-encryption schemes that is called *graded encoding*) to protect against such attacks. Using these tricks, we were able to show in work with Garg, Kalai, Paneth and Sahai [BGK$^+$14] (see also [BR14]) that it is not possible to use the $\oplus$, $\otimes$ and $\overset{?}{=}$ operations to break the obfuscation. This still does not rule out the possibility of an attacker using the raw bits of the encoding (which is in fact what is used in the [BGI+01] impossibility result) but it is a promising sign.

4 "Post modern" cryptography

So far I have avoided all discussion of the *security* of homomorphic quasi-encryption schemes and obfuscation and indeed their status is significantly subtler than other cryptographic primitives such as encryption and digital signatures. To understand these issues, it is worth while to take a step back and look at the question of security for cryptographic schemes in general. The history of cryptography is littered with the figurative corpses of cryptosystems believed secure and then broken, and sometimes with the actual corpses of people (such as Mary queen of Scots) that have placed their faith in these cryptosystems. But something changed in modern times. In 1976 Diffie and Hellman [DH76] proposed the notion of *public key cryptography* and gave their famous Diffie-Hellman key exchange protocol, which was followed soon after by the RSA cryptosystem [RSA78]. In contrast to cryptosystems such as Enigma, the description of these systems is simple and completely public. Moreover, by being *public key systems*, they give more information to potential attackers, and since they are widely deployed on a scale more massive than ever before, the incentives to break them are much higher. Indeed, it seems reasonable to estimate that the amount of manpower and computer cycles invested in cryptanalysis of these schemes today every year dwarves all the cryptanalytic efforts in pre-1970 human history. And yet (to our knowledge) they remain unbroken.

How can this be? I believe that the answer lies in a fundamental shift from "security through obscurity" to "security through simplicity". To understand this consider the question of how could the relatively young and unknown Diffie and Hellman (and later Rivest, Shamir and Adleman) convince the world that they have constructed a secure public key cryptosystem, an object so paradoxical that most people would have guessed could not exist (and indeed a concept so radical that Merkle's first suggestion of it was rejected as an undergraduate project in a coding theory course). The traditional approach towards establishing something like that was "security through obscurity"— keep all details of the cryptosystem secret and have many people try to cryptanalyze it in-house, in the hope that any weakness would be discovered by them before it is discovered by your adversaries. But this approach was of course not available to Diffie and Hellman, working by themselves without much resources, and publishing in the open literature.

Of course the best way would have been to prove a mathematical theorem that breaking their system would necessarily take a huge number of operations. Thanks to the works of Church, Turing and Gödel, we now know that this statement can in fact be phrased as a precise mathematical assertion. However, this assertion would in particular imply that $\mathbf{P} \neq \mathbf{NP}$ and hence proving it seems way beyond our current capabilities. Instead what Diffie and Hellman did (aided by Ralph Merkle and John Gill) was to turn to "security by simplicity"— base their cryptosystem on a simple and well studied mathematical problem, such as inverting modular exponentiation or factoring integers, that has been investigated by mathematicians for ages for reasons having nothing to do with cryptography. More importantly, it is plausible to conjecture that there simply *does not exist* an efficient algorithm to solve these clean well-studied problems, rather than it being the case that such an algorithm has not been found yet due to the problem's cumbersomeness and obscurity. Later papers, such as the pioneering works of Goldwasser and Micali [GM82], turned this into a standard paradigm and ushered in the age of *modern cryptography*, whereby we use precise *definitions* of security for our very intricate and versatile cryptosystems and then *reduce* the assertion that they satisfy these definitions into the conjectured hardness of a handful of very simple and well known mathematical problems.

I wish I could say that the new obfuscation schemes are in fact secure assuming that integer factoring, computing discrete logarithm, or another well-studied problem (such as the LWE problem mentioned in the sidebar) is computationally intractable. Unfortunately, nothing like that is known. At the moment, our only arguments for the security of the constructions of the homomorphic quasi-encryption and indistinguishability obfuscator constructions is that (as of this writing) we do not know how to break them. Since so many potential crypto applications rely on these schemes one could worry that we are entering (to use a somewhat inaccurate and overloaded term) a new age of "post-modern cryptography" where we still have precise definition of security, but need to assume an ever growing family of conjectures to prove that our schemes satisfy those definitions. Indeed, following the initial works of [GGH12, GGH$^+$13] there have been several attacks on their schemes showing limitations on the security notions they satisfy, (e.g., see [CGH$^+$15, Cor15]) and it is not inconceivable that by the time this article is printed they would be broken completely.

While this suggests the possibility that all the edifices built on obfuscation and quasi-encryption could crumble as a house of cards, the ideas behind these constructions seem too beautiful and profound for that to be the case. Once cryptographers have tasted the "promised land" of the great many applications enabled by IO, there is every hope that (as they have so many times before) they would rise to this challenge and manage to construct indistinguishability obfuscators and quasi-encryption based on a single well-studied conjecture, thus placing these objects firmly within the paradigm of modern cryptography. Indeed, this is the focus of an intensive research effort. More than that, one could hope that by following the path these constructions lead us and "going boldly where no man has gone before", we cryptographers will get new and fundamental insights on what is it that separates the easy computational problems from the hard ones.

Thanks to Dan Boneh, Craig Gentry, Omer Paneth, Amit Sahai, Brent Waters and the anonymous CACM reviewers for helpful comments on previous versions of this article. Thanks to Oded Regev for providing me with the figure for the "Learning with Errors" problem.

References

[AHKM14] D. Apon, Y. Huang, J. Katz, and A. J. Malozemoff. Implementing Cryptographic Program Obfuscation. Cryptology ePrint Archive, Report 2014/779, 2014. `http://eprint.iacr.org/`.

[Bar16] B. Barak. Hopes, Fears, and Software Obfuscation. *Communications of the ACM*, 59(3):88–96, 2016. See `http://cacm.acm.org/magazines/2016/3/198855-hopes-fears-and-software-obfuscation/fulltext`.

[BGK$^+$14] B. Barak, S. Garg, Y. T. Kalai, O. Paneth, and A. Sahai. Protecting Obfuscation against Algebraic Attacks. In *EUROCRYPT*, pages 221–238, 2014.

[BGI$^+$01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012. Preliminary version in CRYPTO 2001.

[Bar86] D. A. M. Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. Preliminary version in STOC 1986.

[BSW12] D. Boneh, A. Sahai, and B. Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

[BS02] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003. Preliminary version posted on eprint on 2002, see `https://eprint.iacr.org/2002/080`.

[BR13] Z. Brakerski and G. N. Rothblum. Obfuscating Conjunctions. In *CRYPTO*, pages 416–434, 2013.

[BR14] Z. Brakerski and G. N. Rothblum. Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding. In *TCC*, pages 1–25, 2014.

[CGH$^+$15] J. Coron, C. Gentry, S. Halevi, T. Lepoint, H. K. Maji, E. Miles, M. Raykova, A. Sahai, and M. Tibouchi. Zeroizing Without Low-Level Zeroes: New MMAP Attacks and their Limitations. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 247–266, 2015.

[Cor15] J.-S. Coron. Cryptanalysis of GGH15 Multilinear Maps. Cryptology ePrint Archive, Report 2015/1037, 2015. `http://eprint.iacr.org/`.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[GGH12] S. Garg, C. Gentry, and S. Halevi. Candidate Multilinear Maps from Ideal Lattices. In *EUROCRYPT*, 2013. See also CRyptology ePrint Archive, Report 2012/610.

[GGH$^+$13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *FOCS*, pages 40–49, 2013.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[Gen14] C. Gentry. Computing on the Edge of Chaos: Structure and Randomness in Encrypted Computation. *Proceedings of the 2014 International Congress of Mathematicians (ICM)*, 2014. Also available online at `http://eprint.iacr.org/2014/610`.

[GSW13] C. Gentry, A. Sahai, and B. Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO*, pages 75–92, 2013.

[GM82] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. Preliminary version in STOC 1982.

[Gre14] M. Green. Cryptographic obfuscation and 'unhackable' software, 2014. Blog post. Available at `http://blog.cryptographyengineering.com/2014/02/cryptographic-obfuscation-and.html`.

[JBF03] M. Jacob, D. Boneh, and E. Felten. Attacking an obfuscated cipher by injecting faults. In *Digital Rights Management*, pages 16–31. Springer, 2003.

[Jou00] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. *J. Cryptology*, 17(4):263–276, 2004. Preliminary version in ANTS 2000.

[PRZB12] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009. Preliminary version in STOC 2005.

[RAD78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.