Open questions

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 22 (May 3, 2023)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- One-way functions
- Public-key primitives
- Quantum cryptanalysis
- Secure computation
- What's next?

The plan today is to revisit each of the topics that we covered in the course, and to look at a few of the open research problems in each area. My goals today are to (1) entertain you and (2) convince you that cryptography is one of the most exciting research areas in computer science.

One-way functions, PRGs, PRFs, etc.

We began the class by looking at the one-time pad, a "perfectly secure" one-time cryptosystem, in Shannon's information-theoretic sense. As we saw, the one-time pad—and any perfectly secure one-time encryption scheme—requires a key as long as the message.

One surprise is that it *is* possible to construct information-theoretically secure one-time MACs with short keys. This construction, due to Wegman and Carter [6], inspired the "Galois MAC" that AES-GCM uses.

The revolutionary idea is to base the security of cryptosystems on the conjectured hardness of some computational problem. While cryptographers had implicitly used this idea for centuries, it was not until the field of computational complexity developed—along with the notions of the complexity classes P, NP, etc.—that we could give clean formalizations of these notions.

Early cryptographers never (as far as I know) based the security of their cryptosystems on explicit computational assumptions.

Towards constructing encryption systems with short keys, we defined one-way functions, pseudorandom functions (PRF), pseudorandom permutations (PRP), and pseudorandom generators (PRG). There are a number of deep open questions even about these simple objects. Let me mention just a few, in decreasing order of depth.

In Shannon's era, we essentially classified computational problem as "computable" (e.g., factoring) or "not computable" (e.g., the Halting problem). It makes sense, then, that Shannon's notion of security for a cryptosystem required security in the face of an adversary who could compute any computable function.

Do one-way functions exist?

Most of us believe one-way functions exist, even if we have no idea how to prove it. Recall that

$$(\mathsf{P} = \mathsf{NP}) \Rightarrow \nexists\text{OWF} \qquad \text{therefore} \qquad \exists\text{OWF} \Rightarrow (\mathsf{P} \neq \mathsf{NP}).$$

Even if $\mathsf{P} \neq \mathsf{NP}$, proving that one-way functions exists requires showing something much stronger. More specifically, $\mathsf{P} \neq \mathsf{NP}$ only implies that there *exist* hard instances of certain computational problems (e.g., 3SAT). But to construct a one-way function, we need to come up with a computational problem that is hard *on most inputs* (i.e., "hard on average") and for which we can efficiently sample an instance along with its "solution."

Russell Impagliazzo [5] has a really nice paper on these relationships.

Think of the pair $(x, f(x))$, for a one-way function f, as a solution-problem pair.

Since I do not expect to see a resolution to the P-versus-NP problem any time soon, I really have no hope of seeing a proof that one-way functions exist unconditionally. The strange thing, of course, is that it seems very easy to construct one-way functions: If you cook up any crazy function on n bits, for n large enough, there is a good chance that it will be hard to invert on random inputs. (But if you

have learned anything in this course, I hope that you have learned to not try to cook up your own one-way functions.)

Can we build key exchange from one-way functions?

Another one of the famous open problems in cryptography, that also seems very difficult to solve, is to construct a key-exchange protocol from any one-way function. That is, if I give you a circuit implementing an arbitrary one-way function can you use that circuit to construct a key-exchange protocol that is secure if the underlying one-way function is?

As we have seen in this course, we know how to construct key-exchange protocols from the hardness of the Computational Diffie-Hellman problem, RSA, factoring, LWE, and many other assumptions. But can we construct a key-exchange protocol whose security is based only on the hardness of inverting a one-way hash function?

As I mentioned at the start of the course, Ralph Merkle was an undergrad when he defined the notion of public key exchange and constructed this protocol.

Lest you think that this is a totally absurd idea, let me present Ralph Merkle's key-exchange protocol based on hash functions. The catch is that Merkle's protocol runs in time $O(n)$ for the honest parties and the best attack runs in time $O(n^2)$, on security parameter n.

In contrast, the best attack on standard Diffie-Hellman key exchange in elliptic-curve groups runs in exponential time, roughly $2^{n/2}$.

We use the convenient notation $[x] = \{1, \ldots, x\}$.

Setup. Alice and Bob share a public hash function $H\colon [n^2] \to [n^{10}]$, which we model as a random oracle.

Merkle's Key-Exchange Protocol.

1. Alice picks n numbers $a_1, \ldots, a_n \xleftarrow{R} [n^2]$.
   She sends $H(a_1), \ldots, H(a_n)$ to Bob.

2. Bob picks n numbers $b_1, \ldots, b_n \xleftarrow{R} [n^2]$.
   He sends $H(b_1), \ldots, H(b_n)$ to Alice.

3. Alice and Bob find the least $i, j \in [n]$ such that

   $$H(a_i) = H(b_j).$$

   (If no such pair exists, restart the protocol.) They use $k = a_i = b_j$ as their shared secret.

Actually, they would hash k with an independent hash function and use the result as their shared secret.

Correctness. By the Birthday Paradox, Alice and Bob will agree on a shared secret with good probability: they are each sampling n numbers from $\{1, \ldots, n^2\}$.

Security. We will not make the argument formal, but we can indeed prove security if we model the hash function H as a random oracle.

In particular, any attacker that makes $o(n^2)$ queries to the hash function H has a negligible chance of recovering Alice and Bob's shared secret.

It is not possible to build a better-than-Merkle key-exchange protocol from just a random oracle [1]. But it could be possible to build a key-exchange protocol that makes "non-black-box" use of a one-way function (doesn't just treat the one-way function as a black box but somehow exploits its representation of a circuit, etc.) Such a construction would have to be clever in new ways.

Are there better algorithms for function inversion with preprocessing?

Another one of my favorite open questions has to do with *preprocessing attacks* on one-way functions. This problem still seems hard, but I can believe that a mere moral can solve it. (The first two problems, on the other hand...)

In function inversion with preprocessing, there is an attacker who is trying to invert a function $f\colon [N] \to [N]$. Think of f as as the function:

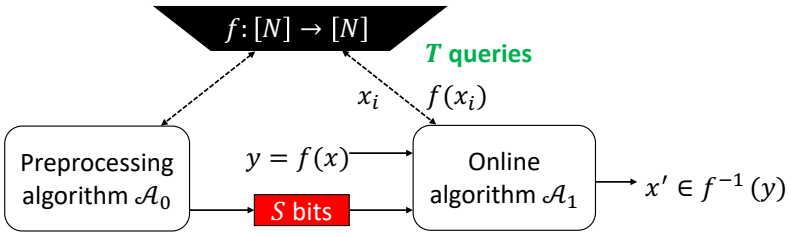
$$f_{\text{AES}}(k) := \text{AES}(k, 00000).$$

So, given just the encryption of all zeros under a secret AES key k, your task is to recover k. If we are given only *oracle access* to f—i.e., we only get to evaluate $f(\cdot)$ in the forward direction and not inspect its representation as a circuit—the best function-inversion algorithm is brute-force search: try all $N = 2^{128}$ AES keys and see which one works.

Martin Hellman [3] first discussed this type of preprocessing attack, in the context of the DES block cipher.

However, it might be the case that our attacker wants to invert f many times on many independent inputs: a powerful government might want to perform AES key recovery many many times. In addition, the attacker might be willing to perform a gigantic amount of *precomputation* to produce a data structure to make the inversion task easier.

The picture looks like this:



This figure is from a talk given by Dima Kogan on our work on function inversion [2].

Now we measure the cost of a function-inversion algorithm by two quantities:

- the **time**—the number of times the adversary evaluates the function f, and
- the **space**—the size of the adversary's precomputed data structure.

The main question in this area is: what is the best achievable trade-off between the space usage S and running time T?

A few points on the trade-off curve are immediate:

- **Brute-force search:** time $T = N$, space $S = 0$.
- **Precompute f^{-1} for all possible inputs:** time $T = 0$, space $S \approx N$.

A good puzzle is to try to come up with an algorithm that achieves $S = T = o(N)$.

Recall that a function $t(N)$ is $o(N)$ if $t(N)/N \to 0$ as $N \to \infty$.

Hellman gave a very surprising algorithm that achieves $S = T = N^{2/3}\,\mathrm{polylog}(N)$. Since Hellman's algorithm only needs to evaluate f in the forward direction, it can invert *any* function f! So it is quite a general result.

[Draw picture of Hellman's algorithm for permutations.]

But are better-than-Hellman preprocessing algorithms possible?

Constructing the data structure would take an epic amount of computation, so it's still not clear that this would have any meaningful real-world consequences for AES.

We know that there is no function-inversion algorithm that simultaneously uses less than $N^{1/2}$ time and $N^{1/2}$ space. But we have no algorithm that achieves $S = T = N^{1/2}$ either. This algorithm would be particularly interesting in the context of AES, since it would give a data structure of size roughly 2^{64} that an attacker could use to break AES in time roughly 2^{64}.

Identification protocols and signature schemes

In the second part of the course, we covered authentication schemes and digital signatures. Let me mention a few interesting open research directions here.

Is there a human-computable many-time-secure authentication protocol?

Say that you get stuck in the dessert in some faraway country hundreds of miles away from the nearest ATM. (This actually happened to me one time.) You call your parents and ask them to wire you $100 via Western Union—which to my surprise works even in places without ATMs. But since your parents are cryptographers, they they want to authenticate you first.

In particular, you share a secret key k with your parents. They will read you a challenge message over the phone and you have to respond, using only a paper and pencil. We want many-time security

against eavesdropping attacks: even if an attacker listens to many interactions between you and your parents, she shouldn't be able to impersonate you.

If you had a PRF F, it would be easy to construct such an authentication protocol: your parents send a nonce r, you reply with $a \leftarrow F(k,r)$, and your parents check that $a = F(k,r)$. So an equivalent question is whether it is possible to construct a PRF (or even a weak variant of a PRF) that a human can compute.

The question, which I suspect Manuel Blum first posed, is whether it is possible to construct an authentication scheme that a human can implement with paper and pencil. What I like about this question is that it first requires you to understand what humans can and cannot compute. Just as we cannot talk about one-way functions without having good models of machine computation, we cannot hope to talk about human-computable one-way functions without having a good model of human computation.

Nick Hopper and Manuel Blum [4] came up with a very simple protocol that is plausibly human computable, and is based on the (very solid) learning-parities-with-noise assumption (LPN).

Hopper-Blum authentication protocol. The two parties shared secret is a vector $k \in \mathbb{Z}_2^n$. Think of $n \approx 4\,096$ or so.

- **Challenge.** The challenge is a random vector $r \xleftarrow{R} \mathbb{Z}_2^n$.
- **Response.** The response is the value

$$a \leftarrow \langle k, r \rangle + \epsilon \qquad \in \mathbb{Z}_2,$$

  where $\langle k, r \rangle = \sum_{i=1}^{n} k_i \cdot r_i \in \mathbb{Z}_2$, and $\epsilon \in \{0, 1\}$ is a biased coin that is 1 with probability $1/10$.
- **Accept/reject.** The verifier accepts if $a = \langle k, r \rangle \in \mathbb{Z}_2$.

If the verifier runs the protocol 1 000 times and accepts if and only if the prover (person authenticating) gives the correct answer at least 80% of the time, then the verifier can be pretty sure that they are talking to the right person.

Is this protocol really human computable? What would a better protocol look like? And what does it mean for a function to be "human computable" or not?

Short signatures

A very simple open problem is: Are there digital signature schemes (under plausible assumptions) in which the signatures are λ bits long and the best forgery attack runs in time very close to 2^λ? We have signatures from pairings/bilinear maps that have bitlength roughly 2λ that achieve λ-bit security, but getting even shorter signatures would be very useful. ECDSA signatures are closer to 3λ bits long, or roughly 384 bits when we are aiming for 128-bit security.

Is the RSA problem as hard as factoring?

Recall that the RSA problem is the problem of taking cube roots modulo the product of two large primes (such that 3 does not divide $p-1$). Is breaking RSA as hard as factoring? In other words: if I give you an efficient algorithm $\mathcal{A}$ for solving the RSA problem, can you use $\mathcal{A}$ to factor the modulus?

Since RSA is going out of style, I am not sure that anyone is working very actively on this sort of problem.

We know that taking *square roots* modulo a composite is as hard as factoring, but for cube roots, and for many other related computational problems modulo composites, we have no idea.

The fact that computing square roots is as hard as factoring is actually based on the identity that you may have learned in high school: $x^2 - y^2 = (x+y)(x-y)$.

How hard really is factoring?

This talk of RSA reminds me of one of the most basic open questions in computational number theory: how hard is factoring? Is there a polynomial-time algorithm for it? The best algorithms today for factoring an n-bit number run in time roughly $2^{\sqrt[3]{n}}$, which is much much better than 2^n, but is much worse than n^{100}, since the exponent grows without bound. It seems unlikely that $2^{\sqrt[3]{n}}$ is the correct running time for the best factoring algorithm in any just world, but the standard tricks cannot do better than that.

Post-quantum signatures

We mentioned a few times over the course of the semester that NIST is in the middle of standardizing a new suite of cryptographic algorithms that are plausibly resistant to attack by quantum computers. Many of the algorithms are either based on hash functions or on lattice problems, similar to the learning-with-errors problem we discussed in the last lecture.

One of the most practically relevant open questions in that area is: can we construct a plausibly post-quantum-secure key-exchange scheme whose communication cost matches that of the Diffie-Hellman protocol. The learning-with-errors-based schemes have much much higher communication costs: the public keys can be megabytes in size, versus 32 bytes for an elliptic-curve Diffie-Hellman public key.

To give a sense of why we need new algorithms in the post-quantum era, let me sketch the two most important results in quantum cryptanalysis:

Grover search. Grover's algorithm solves the following problem:

- **Given:** Oracle access to $f\colon [N] \to \{0,1\}$.
- **Find:** An $x^* \in [N]$ such that $f(x^*) = 1$.

A classical computer requires $\Omega(N)$ queries to f to solve this problem—again assuming that the computer just evaluates f in the forward direction. Grover's result is that a quantum computer can solve this problem using just $O(\sqrt{N})$ queries to f. So, breaking the AES-128 block cipher on a quantum computer would potentially take only time $\approx 2^{64}$.

Notice though that Grover's algorithm only gives a polynomial speedup over a classical algorithm. So the 2^t-time classical algorithm becomes a $2^{t/2}$-time quantum algorithm. To defend, say, AES against Grover search, we just need to double the lengths of our secret keys.

Shor's algorithm. Shor's algorithm, building on an earlier algorithm due to Simon, solves the following problem ("order finding"):

- **Given:** Oracle access to $f\colon [N] \to [N]$.
- **Find:** A non-zero $\Delta \in [N]$ such that $f(x) = f(x+\Delta)$.

On a classical computer, the best algorithm that makes black-box use of f makes at least $\Omega(N)$ queries to f. The shock of Shor's algorithm is that it solves this problem on a quantum computer using only $\text{polylog}(N)$ queries to the function f. Furthermore, it gives quantum polynomial time algorithms for *both* of the widely used hardness assumptions: factoring and discrete log.

I will say a few words about the fallout if I have time.

Like Grover's algorithm, Shor's algorithm works for any function f, so it is extremely general. In particular, it gives a quantum polynomial time attack on discrete log in *every* group.

The next question is how we use Shor's algorithm to break cryptosystems.

- *Discrete log.* Given a group $G = \{g, g^2, g^3, \ldots, g^q\}$ of prime order q, and an instance $R = g^r \in \mathbb{G}$, we want to find $x \in \mathbb{Z}_q$. Define the function

$$F_{g,R}(x,y) := g^x R^y \in \mathbb{G}.$$

  Shor's algorithm gives us back non-zero $(\Delta_x, \Delta_y) \in \mathbb{Z}_q^2$ such that

We are eliding many details here – we need $\Delta_y \neq 0$ in particular.

$$\begin{aligned} g^x R^y &= g^{x+\Delta_x} R^{y+\Delta_y} && \in \mathbb{G} \\ g^x g^{ry} &= g^{x+\Delta_x} g^{ry+r\Delta_y} && \in \mathbb{G} \\ x + ry &= x + \Delta_x + ry + r\Delta_y && \in \mathbb{Z}_q \\ 0 &= \Delta_x + r\Delta_y && \in \mathbb{Z}_q, \end{aligned}$$

  and then we have that $r = -\Delta_x/\Delta_y \in \mathbb{Z}_q$.

Here $1/\Delta_y \in \mathbb{Z}_q$ denotes the unique $\hat{y} \in \mathbb{Z}_q$ such that $y\hat{y} = 1 \in \mathbb{Z}_q$.

- *Factoring.* Given a number $N = pq$, for primes p and q, our task is to factor N. Define the function:

$$F_N(x) := a^x \mod N,$$

where $a \in \mathbb{Z}_N$ is just a random value we pick and fix before we run the algorithm. When working modulo a prime p, we have that

$$a^{x+(p-1)} = a^x(a^{p-1}) = a^x \mod p,$$

where $a^{p-1} = 1 \mod p$ by Fermat's Little Theorem. When the modulus $N = pq$ is composite, the generalization of this rule is:

In other words, the group $\mathbb{Z}_p^*$ has order $p-1$. The group $\mathbb{Z}_N^*$ has order $\phi(N) = (p-1)(q-1)$.

$$a^{x+(p-1)(q-1)} = a^x \mod pq.$$

The quantity $\phi(N) = (p-1)(q-1)$ is called "Euler's totient function," and gives the order of the group of $\mathbb{Z}_N^*$ when N is the product of two primes. (There are generalizations to all composites.)

Now, if we run Shor's algorithm on $F_N(\cdot)$, it will give us back a non-zero integer Δ such that

We could also get a multiple of $\phi(N)$, but that is just as good for the purposes of factoring.

$$a^x = a^{x+\Delta} \mod N.$$

By the discussion that we just had, $\Delta = \phi(N) = (p-1)(q-1)$.

To factor N given $\phi(N)$, we first recover $p+q$ as:

$$\phi(N) = pq - p - q + 1$$
$$N - \phi(N) - 1 = p + q.$$

Now, define the polynomial over the integers:

$$Q(x) := x^2 - (p+q) + N.$$

This polynomial has two roots over the integers: p and q. Finding roots of polynomials over the integers is easy, so we are done.

Building a quantum computer large enough to run either of these algorithms seems very challenging. Depending on who you ask, it is either a mere engineering task or fundamentally impossible. Governments worry about quantum cryptanalysis because an attacker who intercepts your traffic today can potentially decrypt it 50 years from now using a quantum computer. So it's important to "future proof" your cryptosystems for this reason.

Secure computation

The first two thirds of this class focused on how to protect information *in transit*. At this point, we have very solid techniques for encrypting and authenticating data as it flows over a network.

But protecting data in transit only solves part of the problem: Today, it's possible to build a strong encrypted pipe between your computer and Google's servers, but if you still send all of your sensitive to Google, we still have a serious security/privacy problem.

In the last portion of this class, we discussed techniques for securing *computation*: fully homomorphic encryption, delegation of computation, etc. The state of the art in these flavors of secure computation is pretty pitiful, compared to the state of the art in transport encryption.

A few of the problems in this area that I think are most interesting:

- Can you search for something on Google without Google seeing what you're Googling for?
- Can you browse the web without anyone knowing what pages you're browsing to?
- Can you build a trustworthy computer from untrustworthy components?

What next?

- **Attend seminars.** We have a weekly Cryptography and Information Security seminar, a weekly security seminar, and the quarterly Charles River Crypto Day. All are free to attend and open to the public. Most of these events appear on the CSAIL calendar.
- **Take classes.** Related classes at MIT are 6.858 (Spring) and 6.875 (Fall). Yael is teaching a class on proof systems in the fall as well—course number TBD. There are many classes at Harvard relating to privacy, law, cryptography, etc. You can cross-register for these for free!
- **Get involved in research.** For undergraduates: UROPs are a great way to get started, if you haven't tried them already. If you have done a great project in this class, you can share it with your prospective research advisor. The Cryptography ePrint Archive is a place to get the latest research papers in cryptography (for free!).

Closing notes

There are two major themes that have come up again and again in the course this semester.

- The first is the importance of precisely **defining your security goal**. What does it mean for your system to be "secure?"

Against which class of attackers is your system trying to defend? Many of the great innovations in cryptography in the last fifty years have come from thinking through this type of question: the definitions of semantic security, zero knowledge, interactive proofs, digital signatures, and so on.

Many of the most severe security failures come from not having thought through these questions carefully. If you don't even know what it means for your system to be "secure," how can it possibly be?

- The second is that it's often possible to **overcome impossibility**. Almost everything in cryptography seems impossible at first glance:
  - It's impossible to construct a perfectly secure encryption system with a short key.
  - It's impossible to agree on a shared secret without a secret channel.
  - It's impossible to give someone the solution to a problem without revealing the solution to them.
  - It's impossible to compute on data without seeing it.

  And yet, we have seen again that—with clever ideas and the computational assumptions—all of these things are possible!

I hope that you take these two lessons away with you from this course:

1. If you don't know what X means, it's hard to achieve X. In this class $X =$ security, but later on it might be that $X =$ success or $X =$ fulfillment, or $X =$ happiness$\}$. I think the same lesson applies.
2. It is surprisingly often possible, with enough creativity, to overcome barriers. Cryptography is all about turning failures (e.g., our inability to find good factoring algorithms) into success (encryption, signatures, and much more). Maybe the same principle will apply in your life?

Feel free to email me or Yael any time if you have questions about cryptography or anything else. We do not always have capacity to take on UROP/MEng projects, but we're always happy to brainstorm about research ideas and chat about whatever else is on your mind.

Have a great summer!

References

[1] Boaz Barak and Mohammad Mahmoody-Ghidary. Merkle puzzles are optimal–an $o(n^2)$-query attack on any key exchange from a random oracle. In *CRYPTO*, 2009.

[2] Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In *Theory of Cryptography Conference*, 2019.

[3] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[4] Nicholas J Hopper and Manuel Blum. Secure human identification protocols. In *ASIACRYPT*, 2001.

[5] Russell Impagliazzo. A personal view of average-case complexity. In *IEEE Conference on Structure in Complexity Theory*, pages 134–147. IEEE, 1995.

[6] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.