CS 355 NOTES

ARUN DEBRAY
MAY 15, 2014

Contents

1. Pseudorandomness and the Blum-Micali generator: 4/1/14

"This room is really hot. I'm going to come next time in my bathing suit, I think."

There will be three homework assignments in this class (though that sometimes means two, the professor will try to make three this quarter). Please submit them electronically. There are no exams.

The course website is `http://crypto.stanford.edu/~dabo/cs355/`; this contains all of the relevant papers as well as the draft of a textbook (which is password-protected).

We will start with pseudorandomness, which is a foundation and important concept within crypto, but also a good, gentle way to introduce crypto-style proofs.

The first thing we'll do is introduce some notation from crypto and discrete probability, to make sure we're all on the same page.

Definition.

- A probability space $(\Omega, \Pr)$ is a tuple where Ω is a set (which in this class will always be finite) and $\Pr : 2^\Omega \to [0,1]$ such that $\Pr[\Omega] = 1$ and for all sets $F \subseteq \Omega$, the probability of F can be defined as
$$\Pr[F] = \sum_{x\in F} \Pr[x].$$
F is called an event, and is said to happen with probability $\Pr[A]$.
- Two events A and B are independent if $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$.

Definition. There are two kinds of random variables:

- A Type I random variable is a function $X : \Omega \to S$, for a finite set S. For a predicate $Q : S \to \{0,1\}$, we can define
$$\Pr[Q(X)] = \sum_{y\in\Omega} \Pr[y].$$
For example, we could have a random variable that, on the probability space of strings of a given length, returns the sum of their bits, and the predicate could be whether they are even.
  - The notation $X \overset{\text{R}}{\leftarrow} \Omega$ means that X is a uniform random variable on Ω.

  - If A is a randomized algorithm,[1] $A(y;r)$ denotes what A does on input y and randomness r (a source of random bits, in some sense). Then, $X \overset{\text{R}}{\leftarrow} A(y;r)$ indicates that X is the random variable on $\{0,1\}^n$ that is the output of A on input y.
  - For a predicate Q,
$$\Pr\left[X \overset{\text{R}}{\leftarrow} A(y) : Q(X)\right]$$
is the probability that $Q(X)$ holds on an output of A run on input y. For example,
$$\Pr\left[X \overset{\text{R}}{\leftarrow} \{0,1,\dots,n-1\} : X \text{ is even}\right] = \frac{1}{2}.$$

  These are random variables onto a set without structure.
- Type II random variables are functions $X : \Omega \to \mathbb{R}$. These are the familiar random variables from probability classes, allowing one to define $E[x]$, $\operatorname{var}(x)$, $\sigma(x)$, and so on. In particular, we have Chebyshev bounds for estimating inequalities for probabilities.

Now, for some cryptography. A starting point will be pseudorandom generators: definitions, some constructions, and some security.

Definition. A pseudorandom generator (PRG) is an efficiently[2] computable function $G : \{0,1\}^s \to \{0,1\}^s$, where $n > s$. $\{0,1\}^s$ is called the seed space, and $\{0,1\}^n$ is the output space.

What does it mean for a PRG to be secure? These things have been studied forever, and the old definitions (e.g. from Knuth's book) look something like this:

Definition (Knuth)**.** A string X is random if it passes the following twenty statistical tests:

(1) The difference between the number of zeroes in X and the number of ones is small:
$$|\#0(X) - \#1(X)| < \sqrt{n}\log^2 n.$$
(2) The longest run of zeroes in X is at most $\log_2 n \log\log n$.
(3) And so on...

But this definition isn't that great, because an attacker isn't limited to these tests. Here's a better definition.

Definition (Yao, 1982)**.** A PRG G is secure if no efficient statistical test can distinguish the two distributions
$$\underbrace{\left[S \overset{\text{R}}{\leftarrow} \{0,1\}^s : G(s)\right]}_{\text{pseudorandom distribution}} \qquad \text{and} \qquad \underbrace{\left[r \overset{\text{R}}{\leftarrow} \{0,1\}^n\right]}_{\text{random distribution}}.$$

In some sense, instead of twenty tests, we're looking at all possible tests.

Definition. Formally, for an algorithm A, let
$$w_0 \coloneqq \Pr[s \leftarrow \{0,1\}^s, r \leftarrow G(s) : A(r) = 1]$$
and
$$w_1 \coloneqq \Pr\left[r \overset{\text{R}}{\leftarrow} \{0,1\}^n : A(r) = 1\right].$$
Then, the advantage is $\mathrm{PRG}^{(\mathrm{adv})}_{A,G} = |w_0 - w_1|$.

This convention (zero is pseudorandom, one is random) is standard; stick to it for homework and papers.

Now, Yao's definition means that G is a secure PRG if for all efficient algorithms A, $\mathrm{PRG}^{(\mathrm{adv})}_{A,G}$ is negligible (which here means less than $1/\lambda^{\omega(1)}$, but it's easier to just deal with this in one's head to avoid making the notation too messy). For example, $1/2^\lambda$ is neglible, but $1/\lambda^2$ isn't. Basically, the goal is to show that it's less than any polynomial eventually.

Suppose $G : \{0,1\}^s \to \{0,1\}^n$ is secure, and let $\widehat{G} : \{0,1\}^{s+1} \to \{0,1\}^n$ has $\widehat{G}(b \,\|\, s) = b \,\|\, G(s)$, which isn't necessarily secure, e.g. if the seed is always chosen to start with zero. This is a construction that looks very much like homework problems. It's a "proof by sabotage," in which one takes a good generator and sabotages it, which is a great way to provide counterexamples.

[1]The notation of a randomized algorithm can be formalized from a two-tape Turing machine, one tape of which is input, and one tape of which is randomness.

[2]This means polynomial time in this class, but this is a concrete function and thus polynomial running time doesn't strictly make sense. But here, it should mean that there is a *security parameter* $\lambda \in \mathbb{Z}^+$ so that the generator, seed space, and output space are parameterized by λ (i.e. $\{0,1\}^{s(\lambda)}$ and so on), and that the resulting sequence of functions is asymptotically polynomial in λ. This is important but tedious, and thus will be implicit in all other efficient algorithms.

Note at this point that we can't prove that *any* PRGs exist; in fact, to prove that something is a PRG would imply that $\mathbf{P} \neq \mathbf{NP}$, and conversely. That would be neat, but we probably won't get to it in this course.

Definition. A one-bit PRG is a secure PRG $G : \{0,1\}^s \to \{0,1\}^{s+1}$.

Today we're going to assume that one-bit PRGs exist. (It's a nice theorem that the existence of one-way permutations exist implies the existence of one-bit PRGs, but that's a story for next lecture.) Then, we want to produce a secure PRG G_{BM} that takes a one-bit PRG and expands it into a many-bit PRG, using a construction called the Blum-Micali generator.

We'll generalize the notation to $G : S \to R \times S$, where S is the seed space. For a one-bit PRG, $S = \{0,1\}^s$ and $R = \{0,1\}$. Then, the goal is to produce a secure PRG $G_{\mathrm{BM}} : S \to R^n \times S$ for any $n > 0$.

We will do this as follows: G_{BM} takes some $s_0 \in S$ as input. Then for $i = 1, \ldots, n$, let $(s_i, r_i) \leftarrow G(s_{i-1})$; then, output $(r_1, r_2, r_3, \ldots, r_n, s_n) \in R^n \times S$. Here's a picture:

$$\begin{tikzcd}
{} \arrow[r, "s_0"] & \boxed{G} \arrow[r, "s_1"] \arrow[d, "r_1"] & \boxed{G} \arrow[r, "s_2"] \arrow[d, "r_2"] & \cdots \arrow[r, "s_{n-1}"] & \boxed{G} \arrow[r, "s_n"] \arrow[d, "r_n"] & {} \\
& {} & {} & & {} &
\end{tikzcd}$$

Example 1.1. The Blum-Blum-Schub PRG, which is secure if factoring is hard, sets

$$G_N(s) = [s^2 \bmod N, \operatorname{lsb}(s)],$$

where $\operatorname{lsb}(s)$ is the least significant bit function. This is a function $G : \mathbb{Z}_N \to \mathbb{Z}_N \times \{0,1\}$. This generator is actually useless, because it's way too slow; squaring is too slow to only get one bit.

Theorem 1.2 (Blum-Micali)**.** *For any PRG adversary (i.e. efficient randomized algorithms that can distinguish from random) A for G_{BM}, there exists a PRG adversary B for G that runs in polynomial time of A and such that*

$$\mathrm{PRG}^{(\mathrm{adv})}_{A,G_{\mathrm{BM}}} \leq h \cdot \mathrm{PRG}^{(\mathrm{adv})}_{B,G}.$$

Proof. Let A be an adversary for G_{BM}; the goal will to be to torture it by playing a bunch of games, so to speak.

Define $n+2$ distributions P_j on $R^n \times S$ as follows: for each $j = 0, \ldots, n+1$, choose the first j bits of output at random, and then the remaining bits via the Blum-Micali generator. Formally, let $r_1, \ldots, r_j \overset{\mathrm{R}}{\leftarrow} R$ and $s_j \overset{\mathrm{R}}{\leftarrow} S$. and for $i = j+1, \ldots, n+$, let $(s_i, r_i) = G_{\mathrm{BM}}(s_{i-1})$. Then, output $(r_1, r_2, \ldots, r_n, s_n)$. This is sometimes called a hybrid distribution; it is useful for the proof, but not in the real world.

Now, let

$$p_j \coloneqq \Pr\left[z \overset{\mathrm{R}}{\leftarrow} P_j : A(z) = 1\right].$$

Then, $p_0 = w_0$ and $p_n = w_1$, and by the definition of an advantage, $|p_0 - p_n| = \mathrm{PRG}^{(\mathrm{adv})}_{A,G_{\mathrm{BM}}}$. But we can rewrite this as a telescoping sum:

$$|p_0 - p_n| = \left|\sum_{j=0}^{n-1} (p_j - p_{j+1})\right| \leq \sum_{j=0}^{n-1} |p_j - p_{j+1}|,$$

which is just the triangle inequality. Thus, there exists some j^* such that

$$|p_{j^*} - p_{j^*+1}| \geq \frac{1}{n}|p_0 - p_n|.$$

This is exactly because something must be at least as much as the mean. But at the point j^* in the loop, the input is truly random and the output is pseudorandom... and the adversary can distinguish the two distributions, which provides intuition for the construction of B, which will be provided next lecture.

Another way to intuit this is that in the entirely pseudorandom distribution P_0, in which $r_1, \ldots, r_n, s_n$ are all pseudorandom. But for the next distribution P_1, the first bit $\widehat{r}_1$ is chosen randomly, and everything else is pseudorandom. But since G is secure, one ought not to be able to tell the difference; we're playing the secure PRG card, so to speak. But then the next bit is flipped over for P_2, to be truly random instead of pseudorandom, and so on, until finally, at the end, we get to P_n, where everything is random. In some sense, the fact that G is a one-bit PRG, then one can convert pseudorandomness to true randomness in a chain, and no link can be broken (because then the adversary could break the secure G), and thus the entire chain can't be broken: the adversary can't distinguish P_0 from P_n.

2. Proof of Security of the Blum-Micali Generator: 4/3/14

"So first, I'm impressed that you all came back. Congratulations."

The first thing we need to do is provide a full proof of Theorem 1.2, rather than just intuition, but we can also generalize it slightly.

Let (Ω, P_0) and (Ω, P_1) be two distributions over Ω. Then, for any algorithm A, define

$$\mathrm{Adv}_{A,(P_0,P_1)} \coloneqq \left|\Pr\left[X \leftarrow P_0 : A(x) = 1\right] - \Pr\left[x \leftarrow P_1 : A(x) = 1\right]\right|.$$

Definition. P_0 and P_1 are computationally indistinguishable, denoted $P_0 \overset{\mathrm{c}}{\approx} P_1$, if for all efficient algorithms A, $\mathrm{Adv}_{A,(P_0,P_1)}$ is negligible.

Definition. A PRG $G : S \to R$ is secure if

$$\left\{s \overset{\mathrm{R}}{\leftarrow} S, r \leftarrow G(s), \text{ output } r\right\} \overset{\mathrm{c}}{\approx} \left\{r \overset{\mathrm{R}}{\leftarrow} R, \text{ output } r\right\}.$$

Definition. The statistical distance of the two distributions P_0 and P_1 is

$$\Delta(P_0; P_1) = \frac{1}{2}\sum_{x\in\Omega} \left|P_0(x) - P_1(x)\right|.$$

Notice that by the triangle inequality,

$$0 \le \Delta(P_0; P_1) \le \frac{1}{2}\left(\sum_{x\in\Omega}|P_0(x)| + \sum_{x\in\Omega}|P_1(x)|\right) = 1.$$

There are distributions with statistical difference 1, such as two point distributions (i.e. $\Pr(x) = 1$ and $\Pr(y) = 0$ when $y \ne x$) at different points.

Definition. P_0 and P_1 are statistically indistinguishable, written $P_0 \overset{\mathrm{s}}{\approx} P_1$, if $\Delta(P_0; P_1)$ is negligible.

You might be wondering at this point why this definition uses ℓ^1 distance, rather than ℓ^2 distance. This is because of the following result, which holds not just for efficient algorithms but all possible algorithms.

Theorem 2.1. *For all algorithms A, $\mathrm{Adv}_{A,(P_0,P_1)} \le \Delta(P_0; P_1)$.*

The proof of this is somewhat bogged down in details, but the essentials of it are found in the proof of the following weaker theorem. The full proof is in the textbook.

Theorem 2.2. *For all deterministic algorithms A, $\mathrm{Adv}_{A,(P_0,P_1)} \le 2\cdot\Delta(P_0; P_1)$.*

Proof. Let $S_A = \{x \in \Omega : A(x) = 1\}$.[3] Then,

$$\begin{aligned}
\mathrm{Adv}_{A,(P_0,P_1)} &= \left|P_0(S_A) - P_(S_A)\right| \\
&= \left|\sum_{x\in S_A} P_0(x) - P_1(x)\right| \\
&\le \sum_{x\in S_A} \left|P_0(X) - P_1(x)\right| && \text{by the triangle inequality} \\
&\le \sum_{x\in\Omega} \left|P_0(X) - P_1(x)\right| = 2\Delta(P_0; P_1).
\end{aligned}$$

⊠

In words, once you understand the statistical distance between two statistical distributions, you know how hard it is to computationally distinguish them.

Note that with a bit more analysis, one can get rid of the factor of 2, and that the converse is false (e.g. PRGs, which have considerable statistical distance from the uniform distribution, but are presumably completely indistinguishable).

Example 2.3. Suppose $m > n$, and let

$$\Delta(m; n) \coloneqq \Delta\left(r_0 \overset{\mathrm{R}}{\leftarrow} \{1, \dots, m\}; r_1 \overset{\mathrm{R}}{\leftarrow} \{1, \dots, n\}\right),$$

so that we're comparing the uniform distribution on a set with the uniform distribution on a subset. Then, for $i \le n$, the difference is $1/n - 1/m$, and elsewhere, the difference is $1/m$ (since it's zero for the other distribution). Thus, the sum is

$$\Delta(m; n) = \frac{1}{2}\left(\sum_{x=0}^{n-1}\left(\frac{1}{n} - \frac{1}{m}\right) + \sum_{x=n}^{m-1}\frac{1}{m}\right) = \frac{m-n}{m}.$$

[3]Since A is deterministic, this is in fact a set, rather than a probability distribution, which is why this proof is so much simpler.

Remember this calculation; it's a very important fact in crypto. Notice that this means that when m is close to n, then the two distributions are indistinguishable, especially when n is large. This is a good way to bridge gaps in proofs (e.g. if you can only show something is secure on a very slightly smaller distribution).

For an application, suppose we're given a secure PRG $G : S \to \{0, \dots, 2^\ell - 1\}$, and want to construct a $G' : S \to \{0, 1, \dots, n-1\}$, where $n < 2^\ell$.[4] Define $G'(s) = G(s) \bmod n$. Under what conditions is G' secure?

Theorem 2.4. *For all adversaries A for G', there exists an adversary B for G such that*

$$\mathrm{Adv}_{A,G'}^{(\mathrm{PRG})} \le \mathrm{Adv}_{B,G}^{(\mathrm{PRG})} + \frac{n}{2^\ell},$$

and such that B runs in about the same running time as A.

Thus, for G' to be secure, we need $n/2^\ell$ to be negligible. In practical terms, this means it should be less than $1/2^{128}$, so $\ell \approx \log_2 n + 128$.

Proof of Theorem 2.4. First, for notation, define $m = 2^\ell$ and $k = \lfloor m/n \rfloor$, so that $0 < m - k/n < n$.

Then, given an algorithm A attacking G', construct an algorithm B which, on input $x \in \{0, \dots, m-1\}$ outputs $A(x \bmod n)$. So why does this break G? Let

$$p_0 \coloneqq \Pr\left[s \xleftarrow{\mathrm{R}} S, r \leftarrow G(S) \bmod n : A(r) = 1\right] = \Pr\left[s \xleftarrow{\mathrm{R}} S, r \leftarrow G(s) : B(r) = 1\right],$$

which is just notation-chasing. And the second term of the advantage is

$$\begin{aligned} p_1 &\coloneqq \Pr\left[r \xleftarrow{\mathrm{R}} \{0, \dots, n-1\} : A(r) = 1\right] \\ &= \Pr\left[r \xleftarrow{\mathrm{R}} \{0, \dots, k \cdot n - 1\} : A(r \bmod n) = 1\right] \\ &= \Pr\left[r \xleftarrow{\mathrm{R}} \{0, \dots, kn - 1\} : B(r) = 1\right]. \end{aligned}$$

But this isn't yet in terms of the advantage of B, because it wasn't in terms of the distribution on $\{0, \dots, kn-1\}$. Instead, we can introduce an intermediate distribution which has the probabilities we like. This is a common trick in these sorts of proofs. Thus, let

$$p^* = \Pr\left[r \xleftarrow{\mathrm{R}} \{0, \dots, n-1\} : B(r) = 1\right].$$

Now, we can once again use the triangle inequality:

$$\begin{aligned} \mathrm{Adv}_{A,G'}^{(\mathrm{PRG})} &= |p_1 - p_0| = |p_1 - p^* + p^* - p_0| \\ &\le |p_1 - p^*| + |p^* - p_0| \\ &\le \frac{m - kn}{m} + \mathrm{Adv}_{B,G}^{(\mathrm{PRG})} \\ &\le \frac{n}{m} + \mathrm{Adv}_{B,G}^{(\mathrm{PRG})}. \end{aligned}$$ ⊠

This proof might look somewhat complicated, but it's in some sense trivial; the only trick is to introduce the new distribution, and then use the statistical distance. This sort of proof is sometimes known as a hybrid argument.

Now, back to the Blum-Micali generator, and specifically the proof that the construction is secure (in a specific sense mentioned in the theorem statement).

Proof of Theorem 1.2. Let A be a PRG adversary for G_{BM}, and our goal is to build an algorithm B that breaks G. Furthermore, use the notation that $\widehat{r}$ is random and r is pseudorandom for bits r.

For $j = 0, \dots, n$, pet p_j be the following hybrid distribution: let $\widehat{r}_1, \dots, \widehat{r}_j \xleftarrow{\mathrm{R}} R$ and $s_j \xleftarrow{\mathrm{R}} S$, and then for $i = j+1, \dots, n$, let $(r_i, s_i) \leftarrow G(S_{i-1})$. Then, output $(\widehat{r}_1, \dots, \widehat{r}_j, r_{j+1}, \dots, r_n, s_n)$. Thus, p_0 is entirely pseudorandom and p_n is totally random, and if $p_j \coloneqq \Pr[z \leftarrow P_j : A(z) = 1]$, then $\mathrm{Adv}_{A,G_{\mathrm{BM}}}^{(\mathrm{PRG})} = |p_0 - p_n|$.

The goal is to show that $P_0 \overset{c}{\approx} P_1 \overset{c}{\approx} \cdots \overset{c}{\approx} P_n$, and thus that $P_0 \overset{c}{\approx} P_n$ (since it's an equivalence relation).[5]

The general idea is illustrated by arguing that A can't distinguish P_j and P_{j+1}: specifically, that there exists a B_j such that $|p_j - p_{j-1}| = \mathrm{Adv}_{B_j,G}^{(\mathrm{PRG})}$, which is negligible. In other words, if this algorithm is non-negligible, then B_j can break G.

Specifically, on input $(r^*, s^*) \in R \times S$, B_j does the following: it generates random bits $\widehat{r}_1, \dots, \widehat{r}_j \in R$ and sets $s_{j+1} := s^*$. Then, for $i = j+2, \dots, n$, let $(r_i, s_i) \leftarrow G(s_{i-1})$, and output $A(z)$, where $z = (\widehat{r}_1, \dots, \widehat{r}_j, r^*, r_{j+2}, \dots, r_n, s_n)$.

[4]One place this appears is Diffie-Hellman, since the order of the group is prime and therefore not a power of 2.

[5]This is a common enough technique that for experts in crypto, this would be the end of the proof.

if $(r^*, s^*) \overset{R}{\leftarrow} S$, then $z \overset{R}{\leftarrow} P_{j+1}$, because the first $j+1$ bits of z are truly random, and the remaining bits are pseudorandom, which is exactly what P_{j+1} is. Thus,

$$\Pr\left[(r^*, s^*) \overset{R}{\leftarrow} R \times S : B(r^*, s^*) = 1\right] = p_{j+1}.$$

In the second case, where $(r^*, s^*) \leftarrow G(s)$ where $s \overset{R}{\leftarrow} S$, then $z \overset{R}{\leftarrow} P_j$, because the first j bits are truly random, and the remaining bits are pseudorandom. Thus,

$$\Pr\left[s \overset{R}{\leftarrow} S, (r^*, s^*a) \leftarrow G(s) : B(r^*, s^*) = 1\right] = p_j.$$

Thus,

$$\begin{aligned}\mathrm{Adv}^{(\mathrm{PRG})}_{A, G_{\mathrm{BM}}} &= |p_0 - p_n| = |p_0 - p_1 + p_1 - p_2 + \cdots + P_{n-1} - p_n| \\ &= \sum_{i=0}^{n-1} |p_i - p_{i+1}| \\ &= \sum_{i=0}^{n-1} \mathrm{Adv}_{B_i, G}\,.\end{aligned}$$

But since G is secure, then each of these must be negligible, and therefore their sum is. Yet we have something slightly stronger to prove; if G_{BM} is insecure, we don't know yet which one of the B_i has non-negligible advantage, which is necessary to complete the proof.

So... have the algorithm B choose one of the B_i at random and run it. This breaks G with non-negligible probability, as $\mathrm{Adv}_{B,G} = (1/n) \cdot \mathrm{Adv}_{A, G_{\mathrm{BM}}}$. This involves writing some conditional probabilities, conditioning on j:

$$\Pr[B = 1] = \sum_{k=1}^{n-1} \Pr[B = 1 \mid j = k] \Pr[k] = \sum_{k=1}^{n-1} \Pr[B_j = 1] \frac{1}{n},$$

and then the negligible terms go away. ⊠

This proof seems very tedious, but the point is that all of the details have been done once, and now the proofs in class can be a little more streamlined, with the technical details left to the exercises.

Example 2.5. Blum-Micali generators occur in the real world:

- RC4 was recently declared dead (don't use it!), but it's still used by much of the world to encrypt Web traffic. The state of the generator is an array s of length 255, along with two pointers i and j into the array. Then, it outputs $s[s[i] + s[j]]$, and then increments i by 1 and sets $j \leftarrow j + s[i]$. Then, swap $s[i]$ and $s[j]$. This is viewed as a Blum-Micali generator in that it outputs something, and then a new state.

  The specific issue with RC4 is that it leaks the first few bytes (which often are the authentication cookie, which is bad), so if you must use it, let it run for a little while before beginning encryption.
- There is another Blum-Micali generator called Trivium.
- A bad example, called LCG, the linear combinational generator. Fix a prime p, and $a, b \in \mathbb{Z}_p$. Then given some seed $s \in \mathbb{Z}_p$, let $\mathrm{LCG}(s) = (\mathrm{lsb}(s), as + b \bmod p)$. This is completely insecure, and admits a nice geometrical attack involving lattices: given a few bits of the output (roughly $\log p$), one can reconstruct the seed. Oops.
- Last time, we saw the BBS generator, which changes and fixes this to $\mathrm{BBS}(s) = (\mathrm{lsb}(s), s^2 \bmod N)$, which is as secure as factoring, where $N = pq$ is an RSA modulus.

3. The Goldreich-Levin Theorem: 4/8/14

"It would be the joke of the century if Dan Bernstein turned out to be an agent of the NSA."

Recall last week that we introduced the notion of computational indistinguishability of distributions, $P_0 \overset{c}{\approx} P_1$, and that of statistical indistinguishability $P_0 \overset{s}{\approx} P_1$. Then, we defined a PRG $G : S \to R$ to be secure if $\{s \overset{R}{\leftarrow} S : G(S)\} \overset{c}{\approx} \{r \overset{R}{\leftarrow} R : r\}$. The Blum-Micali generator accepts PRGs:

$$G : S \to R \times S \xrightarrow{\mathrm{BM}} G_{\mathrm{BM}} : S \to R^n \times S.$$

The GGM generator takes a generator that doubles the size of the seed (which can be constructed via Blum-Micali or other means) $G : S \to S \times S$, and will produce a $G^{(d)}_{\mathrm{GGM}} : S \to S^{(2^d)}$. In essence, given some $s \in S$, $G^{(1)} : s \mapsto G(s) = (s_1, s_2)$, then $G^{(2)} : s \mapsto (G(s_1), G(s_2))$, which has four elements, and so on.

Theorem 3.1. *If G is a secure PRG, then for any constant d, $G_{\text{GGM}}^{(d)}$ is also a secure PRG. Specifically, for all efficient adversaries A for $G_{\text{GGM}}^{(d)}$ there exists an adversary B for G such that*

$$\mathrm{Adv}_{A,G_{\text{GGM}}^{(d)}}^{(\text{PRG})} \leq 2^d \,\mathrm{Adv}_{B,G}^{(\text{PRG})}.$$

Proof. Consider the tree of values produced by G_{BBM}: s_1 and s_2 from some random $\widehat{s}$, $s_3, \dots, s_6$ from s_1 and s_2, and so on. But this is computationally indistinguishable from the case where $\widehat{s}_1$ and $\widehat{s}_2$ are chosen by random and plugged into the generator for s_1 and s_2 (for if it were distinguishable, then G wouldn't be secure, as in the proof from last lecture). But this is computationally indistinguishable from the case where $\widehat{s}_3, \dots, \widehat{s}_6$ are truly random rather than pseudorandom, and so on. ⊠

It turns out one can strengthen this construction to produce a PRF.

Of course, nobody uses this generator to actually construct stream ciphers. In the real world, there is exactly one generator used in the real world, and that is counter mode (e.g. one called ChaCha, which is or will be used at Google and was effectively pulled out of a hat by Dan Bernstein). This is based on a PRP.

One cute application of PRGs is to derandomize randomized algorithms. This is related to the major open problem in complexity theory as to whether BPP $\subseteq$ **P**. This almost comes for free with the existence of secure PRGs. Ket $A(x; r)$ be a randomized efficient algorithm for some task (e.g. factoring, or graph coloring), and suppose that for all $x \in I$ (where I is the input space):

$$\Pr\left[x \overset{\text{R}}{\leftarrow} R : A(x; r) = \text{“success”}\right] \geq \frac{1}{3},$$

i.e. for at least a third of the r, A finds a solution to the problem x. The power of randomness is that we know there are many needles in the haystack that work (depending on x, of course), but we don't know ahead of time which ones work, and have to try them.

Let $G : S \to R$ be a secure PRG. Then, introduce a deterministic algorithm $B(x)$ that, for all $s \in S$, tries $A(x; G(s))$, and outputs $G(s)$ if the result is successful, and fails if none succeed. This has running time $\text{time}(B) = \text{time}(A) \times |S|$ (where G is assumed to be efficient), so as long as the size of the seed space is polynomial in that of x, then B is also polynomial-time if A is. Constructions like this are a good reason to keep seed spaces small.

Claim. If G is a secure PRG, then for all $x \in I$, $B(x)$ will never fail.

Proof. Suppose there is an $x \in I$ that makes B fail. This introduces a distinguisher D for G, which breaks the secure generator. D works as follows: given some $r \in R$,

$$D(r) = \begin{cases} \text{“random,”} & \text{if } A(x; r) = \text{“success”} \\ \text{“pseudorandom,”} & \text{otherwise.} \end{cases}$$

Thus,

$$\mathrm{Adv}_{D,G}^{(\text{PRG})} \geq \frac{1}{3} - 0 = \frac{1}{3},$$

(the latter because when x is bad, D never claims it to be pseudorandom), which is certainly non-negligible. ⊠

Notice that this proof depends on x being given to D from somewhere.

Now, this is not a proof that BPP $\subseteq$ **P**; we currently don't know how to prove the existence of a secure PRG (which also forces $\mathbf{P} \neq \mathbf{NP}$ — so this proof isn't a great way to approach the complexity theory, but it's still pretty neat). Since we don't have polynomial-time PRGs, then one might use a partial result, e.g. a PRG that is secure against log-space algorithms, which are weaker.[6] So this proof works absolutely in this restricted case.

The Goldreich-Levin Theorem. This theorem is a classic result in crypto, allowing one to build PRGs from weaker assumptions.

Definition.

(1) A one-to-one function $f : X \to X$ is called a one-way permutation if:
- f is efficiently computable, and
- for all efficient adversaries A for f, $\Pr\left[x \overset{\text{R}}{\leftarrow} X : A(f(x)) = x\right]$ is negligible, i.e. f is “hard to invert.”

(2) A function $f : X \to Y$ is called a one-way function if:
- f is efficiently computable, and
- for all efficient algorithms A for f, $\Pr\left[x \overset{\text{R}}{\leftarrow} X : f(A(f(x))) = f(x)\right]$ is negligible.

[6]Log-space, or L, is a class of deterministic algorithms that are restricted to space logarithmic in the size of the input. L $\subseteq$ **P**.

Examples include cryptographic hash functions, such as SHA-256, $f(x) = \text{AES}(x, 0)$, and so on. RSA is an example of a one-way permutation (which is slow and has lots of structure, and therefore isn't the best first example). By contrast, one-way permutations are easy to construct: we know of no fast one-way permutations. There are some from algebra, though, such as $f : \{1, \dots, p-1\} \to \{1, \dots, p-1\}$ defined as $f(x) = g^x \bmod p$ (which is mathematically ugly, in that it conflates multiplication and addition, but a bijection anyways). RSA is a bit of a problem, because it depends on who knows the factorization; it can be formalized to be an OWP, however.

Even though one-way permutation are hard to construct, much of the theory of building PRGs depends on them.

Theorem 3.2 (Goldreich-Levin)**.** *Given a one-way permutation f, one can build a secure PRG.*

Then, one can use the PRG to build a PRF and therefore a PRP. There is a construction going from OWFs to PRGs, but it's much, much more complicated. The theoretical result is nice, but this theorem will find lots of other uses in this course.

Interestingly, in practice it often goes the other way: one tends to start with a PRP, and then use counter mode to build a PRG, or the switching lemma to build a PRF.

Definition. Suppose $f : X \to Y$ is a one-way function. Then, another function $h : X \to R$ is said to be hardcore for f if

$$\left\{x \xleftarrow{\text{R}} X : (f(x), h(x))\right\} \overset{\text{c}}{\approx} \left\{x \xleftarrow{\text{R}} X, r \xleftarrow{\text{R}} R : (f(x), r)\right\}.$$

In other words, even if $f(x)$ is already given, it's still not possible for an adversary to distinguish $h(x)$ from random.

A typical example is when $R = \{0, 1\}$, in which case h is also called a hardcore bit.

Theorem 3.3. *Let p be prime, $g \in \mathbb{Z}_p^*$ be of prime order q, and $f : \{1, \dots, p-1\} \to \mathbb{Z}_p^*$ sending $x \mapsto g^x$. Then, if f is a one-way function,*[7] *then $h(x) = \text{lsb}(x)$ is hardcore for f.*

Here, $\text{lsb}(x)$ denotes the least significant bit of x. Interestingly, the corresponding $h'(x) = \text{msb}(x)$, the most significant bit, is not hardcore.

In other words, even given exponentiation, it's not possible to distinguish the least significant bit from random. The proof is quite nice: given a distinguisher for only the least significant bit from random allows one to recover the entire discrete log.

For another example, called the BBS hardcore bit, one has the following theorem.

Theorem 3.4. *Suppose $N = pq$ for large primes p and q, and let $f : \mathbb{Z}_N \to \mathbb{Z}_N$ send $x \mapsto x^e$ for some RSA constant $e \neq 2$.*[8] *Then, if f is a one-way permutation, then $h(x) = \text{lsb}(x)$ is a hardcore bit for f.*

Lemma 3.5. *Suppose $f : X \to X$ is a one-way permutation and $h : X \to R$ is hardcore for f. Then, there exists a secure PRG $G : X \to X \times R$ given by $G(s) = (f(s), h(s))$.*

Proof. Since h is hardcore, then

$$\begin{aligned}\left\{s \xleftarrow{\text{R}} X : (f(s), h(s))\right\} &\overset{\text{c}}{\approx} \left\{s \xleftarrow{\text{R}} S, r \xleftarrow{\text{R}} R : (f(s), r)\right\} \\ &\overset{\text{s}}{\approx} \left\{s \xleftarrow{\text{R}} X, r \xleftarrow{\text{R}} R : (s, r)\right\},\end{aligned}$$

the latter equivalence because f is a permutation: if one takes a random element of S and applies f, then the result is another random element of S. ☒

Notice that the last step implies that this construction doesn't work for one-way functions. A typical counterexample is, given a one-way function f, $g(x) = [0 \,\|\, f(x)]$ is still a one-way function, but the construction given by the lemma isn't random.

Now, we can provide two constructions of hardcore bits. The first, the Goldreich-Levin method: suppose $f : \{0,1\}^n \to \{0,1\}^n$ is an OWP, and define $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \to \{0,1\}^n$ as $(x, r) \mapsto (f(x), r)$. Then, the content of Theorem 3.2 is that if f is a one-way permutation, then the inner product

$$B(x, y) = \langle x, r \rangle = \sum_{i=1}^{n} x_i \cdot r_i \bmod 2$$

is a hardcore bit for g. This is also known as the Goldreich-Levin bit.

[7]This is the discrete-log assumption, which is believed to be true but still technically open.

[8]For example, one could choose $e = 3$ as long as $3 \nmid \varphi(N)$.

Proof sketch for Theorem 3.2. Suppose $B(x, r)$ isn't hardcore for g. Then, there exists an efficient algorithm D such that

$$\Pr\left[x, r \to \{0,1\}^n : D(g(x, r)) = B(x, r)\right] \geq \frac{1}{2} + \varepsilon,$$

where ε is some non-negligible function. This follows from a few mechanical steps that involve expanding out the definition of a hardcore bit.

Let

$$S = \left\{x \mid \Pr\left[r \overset{\text{R}}{\leftarrow} \{0,1\}^n : D(f(x), r) = \langle x, r\rangle\right] \geq \frac{1}{2} + \frac{\varepsilon}{2}\right\}.$$

Then, it's a bit more mechanical work (some simple combinatorics) to show that $|S| > (\varepsilon/2) \cdot |X|$.

Now, fix an $x \in S$; then, D can be thought of as a function in just r. Specifically, there exists a D' such that

$$\Pr\left[r \overset{\text{R}}{\leftarrow} \{0,1\}^n : D'(r) = \langle r, x\rangle\right] \geq \frac{1}{2} + \frac{\varepsilon}{2}.$$

Then, a result called the Goldreich-Levin algorithm (which will be presented next lecture) will show that D' can recover all such x with this property, even with this slight advantage. (It turns out there are at most $O(1/\varepsilon^2)$ such x.)

Suppose $\Pr[D'(r) = \langle r, x\rangle] = 1$; then, it's trivial to recover x (in some sense, an extreme case of the algorithm). If instead $\Pr[D'(r) = \langle r, x\rangle] \geq 3/4$, then one can build a $D''(r)$ that is perfect in that it can recover x. This is a good exercise; the general theorem is a little harder.

4. Commitments: 4/10/14

"One of the most important lessons of the Heartbleed bug is that if you ever in the future discover an important exploit in a popular protocol, the first thing you must do is get a really cool logo."

Recall that if $f : X \to X$ is a one-way permutation, then $h : X \to \{0,1\}$ is a hardcore bit for f if

$$\left\{x \overset{\text{R}}{\leftarrow} X : (f(x), h(x)\right\} \overset{\text{c}}{\approx} \left\{x \overset{\text{R}}{\leftarrow} X, b \overset{\text{R}}{\leftarrow} \{0,1\} : (f(x), b)\right\},$$

or equivalently that for all efficient adversaries A,

$$\left|\Pr\left[x \overset{\text{R}}{\leftarrow} X : A(f(x)) = h(x)\right] - \frac{1}{2}\right|$$

is negligible.

We then defined the Goldreich-Levin bit: given a one-way permutation $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$, then we define $g : \mathbb{Z}_2^{2n} \to \mathbb{Z}_2^{2n}$ by $g(x, r) = (f(x), r)$. Then, Theorem 3.2 is that

$$h(x) = \langle x, r\rangle = \sum_{i=0}^{n-1} x_i \cdot r_i \bmod 2$$

is hardcore for g.

The key to this is not so much the theorem statement but the proof idea: if A is an algorithm such that

$$\Pr\left[r \leftarrow \mathbb{Z}_2^n : A(r) = \langle x, r\rangle\right] > \frac{1}{2} + \varepsilon,$$

then there exists an algorithm B that can use A to output a list of $o(1/e^2)$ values (by running A on a bunch of different possible r), where x is one of the values. This is much easier to show when the $1/2$ is replaced with $3/4$.

This algorithm has applications throughout crypto, and even in learning theory and coding theory! A reinterpretation of the Goldreich-Levin algorithm for coding theory is as follows.

The Hadamard code is a set of code words in $\mathbb{Z}_2^n$: specifically, for every $x \in \mathbb{Z}_2^n$, we have a code word with length 2^n (i.e. the code words are in $\mathbb{Z}_2^{2^n}$), in which the i^{th} bit in the code word for x (where $i = 1, \ldots, 2^n$) is $\langle i, x\rangle$. In effect, this computes the parity function for the bits that select for x. For example, if $x = 000$, then the code word for x is 00000000. If $x = 001$, then its code word is 01010101. If $x = 010$, then its code word is 00110011. These sum (mod 2) to the code word for $x = 011$: 01100110. It turns out that any two code words are orthogonal to each other, and the minimal distance in this code (i.e. between any two words) is very large: $2^n/2$. This means that it can detect this many errors (which is a lot, which is good).

We can think of the algorithm A as defining a noisy codeword $(A(000), A(001), \ldots, A(111)) \in \mathbb{Z}_2^{2^n}$ for some x. Then, the Goldreich-Levin algorithm finds all codewords x that match A at $1/2 + \varepsilon$ positions.[9] If $3/4$ of the code is intact, then there is a unique x; in general, there are $o(1/\varepsilon^2)$ such x. In summary, the Goldreich-Levin algorithm is a list decoding algorithm for the Hadamard code.

[9]This procedure is sometimes called list decoding; the idea is that even if almost half of the bits are corrupted, then the original code can be recovered.

There are many generalizations of this; here's one due to Näslund. Let $f : \mathbb{Z}_q \to \{0,1\}^m$ be a one-way function, and let $g(x, a, b) = (f(x), a, b)$, where q is an n-bit prime and $a, b \in \mathbb{Z}_q$.

Theorem 4.1 (Näslund)**.** *If f is injective,*[10] *then for all i such that $1 \le i \le n - \log_2 n$, then*

$$B_i(x, a, b) = (ax + b \bmod q)|_i$$

(i.e. its i^{th} bit) is hardcore for g.

Notice that we have to ignore the highest bits, because they leak information about how far a prime number is from a power of 2. This generalization moves from inner products to arbitrary arithmetic mod q. Then, one gets a very similar algorithm, called Naslund's alorithm, which can recover possible x if

$$\Pr_r [D(r) = (ax + b \bmod q)|_i] \ge \frac{1}{2} + \varepsilon.$$

Corollary 4.2. *if $g \in \mathbb{Z}_p^*$ is of prime order q, then let $G = \langle g \rangle$. Then, most bits of the discrete log is hardcore: for all i with $1 \le i \le n - \log_2 n$, $x|_i$ is hardcore for $f(x) = g^x \bmod p$ on $\{1, \dots, q\} \to G$.*

Proof. Suppose there exists an i with $1 \le i \le n - \log_2 n$ such that there is an algorithm A_i for which

$$\Pr [x \to \{1, \dots, q\} : A(g^x \bmod p) = x|_i] \ge \frac{1}{2} + \varepsilon.$$

This can be used to build an algorithm B such that for any $y \in G$, $B(y) = \mathrm{Dlog}_g y$. Specifically, let $h_y(a, b) = (ax + b \bmod q)|_i$, where $x = \mathrm{Dlog}_g y$.

Note that $y^a g^b = g^{ax+b}$, so $H_y(a, b)$ is the i^{th} bit of $ax + b$, but this is also the i^{th} bit of $\mathrm{Dlog}_g(y^a g^b)$. Then, using A_i,

$$\Pr_{a,b \leftarrow \mathbb{Z}_q} \left[A_i(y^a g^b) = h_y(a, b)\right] > \frac{1}{2} + \varepsilon,$$

and thus we can recover $x = \mathrm{Dlog}_g y$. ⊠

This says that the i^{th} bit of the discrete log is as hard to compute as most of the discrete log, for most i. This is pretty remarkable.

Commitments. A trivial motivation for commitments is coin-flipping over the telephone. The original story is: Alice and Bob are getting a divorce, so Bob calls Alice and asks, "hey, how are we splitting up our assets?" Then, the agree that Alice should flip a coin repeatedly and Bob will get the item if he guesses correctly. Of course, this scheme by default doesn't exactly ensure Bob's trust in Alice, so is there a better, more fair[11] way to handle this?

The idea here is that a sender S has a bit, and then the sender and the receiver execute a protocol (the commit phase) so that R has a commitment. Then, later, S can open the commitment with another protocol, and R can accept or reject according to the protocol.

The correctness property is that if both S and R follow the protocol correctly, then R accepts iff S opens the commitment correctly. But there are two security properties and four definitions:

Definition.

- Suppose R is the adversary, and sends m_0, m_1 to the challenger, and receives commit(m_b) and attempts to guess b' for b (i.e. whether $b = 0$ or $b = 1$). Then, the hiding advantage of a commitment scheme A is

  $$\mathrm{Adv}^{(\mathrm{H})}_{A,(S)} = |\Pr [b' = 1 \mid b = 0] - \Pr [b' = 1 \mid b = 1]| .$$

  Then, a scheme is *perfectly hiding* (PH) if for any unbounded adversary A, its hiding advantage is negligible; it is *computationally hiding* if the advantage is negligible for all efficient algorithms A (e.g. no factoring).
- If instead S is the adversary, and R is the challenger, then R sends a m_0 and m_1 and receives a commitment for one of them. Then, S tries to open both of them (i.e. we check if R would accept the commitment being opened as m_0 and would accept it as m_1). Then, define the binding advantage of a commitment scheme A

  $$\mathrm{Adv}^{(\mathrm{B})}_{A,R} = \Pr[A \text{ can be successfully opened in 2 ways}\}.$$

  Then, a scheme is *perfectly binding* (PB) if for any unbounded adversary A, its binding advantage is negligible; it is *computationally binding* if the advantage is negligible for all efficient algorithms A.

[10] Sometimes, the notation "one-way permutation" is used to mean an injective one-way function, even if the domain and range are't the same.

[11] "Fair" is a problematic word to use here, for a few reasons that we won't get into. We can think of this as both of them learning the answer at the same time.

Hiding security means that Bob can't figure out the committed value before it opens, and binding security means Alice can't open the commitment in two different ways and have Bob accept either.

It's an unfortunate fact that there are no commitments that are both perfectly hiding and perfectly binding, though there are perfectly hiding schemes that are computationally binding (which are useful in zero-knowledge proofs) as well as computationally hiding schemes that are perfectly binding.

For the first construction of such a scheme, which will be computationally hiding, but perfectly binding, let $f : X \to X$ be a one-way permutation (or $f : X \to Y$ be a collision-resistant hash function in one useful variant) and let $B : X \to \{0,1\}$ be a hardcore bit for f. Then, to generate a commitment for a $b \in \{0,1\}$, S chooses an $x \overset{\text{R}}{\leftarrow} X$, lets $y = f(x)$, and $c = b \oplus B(x)$, and sends $[y, c]$ to the receiver. Then, to open, send x and b to R, who accepts iff $y = f(x)$ and $c = b \oplus b(x)$.

This is trivially perfectly binding: no matter how smart S is, there is exactly one x such that $f(x) = y$, and x and c determine b, so R accepts only (x, b). In the case where f is a collision-resistant hash function instead of a one-way permutation, if S can send (x_0, b_0) and (x_1, b_1) where $b_0 \neq b_1$, then $x_0 \neq x_1$ (since x completely determines b), but $f(x_0) = y = f(x_1)$ is a collision for f. Thus, this is computationally binding, which seems slightly worse, but is still reasonable for security and is a much faster algorithm (since one-way functions come from slow algebra).

This scheme is hiding, because b is hardcore: given $f(x)$, b is still uniformly random, so $b \oplus B(x)$ is a one-time pad! But the definition of hardcore implies this is (for both kinds of f) computationally binding.

A couple things to note about this:

- This is what is known as a malleable commitment scheme: given commit(b) and some b', one can compute commit$(b \oplus b')$ (just xoring them). This is a very cool area of research (in terms of computation on commitments or, conversely, on finding non-malleable commitment schemes). Malleable schemes aren't always good; for example, in an auction, Alice might commit some bid n, and then Eve computes on this commitment to obtain a commitment for $n + 1$, thus outbidding Alice without knowing her original bid!
- This commitment scheme looks like symmetric encryption. But don't take the analogy too far, because here's a very bad commitment scheme: suppose S chooses $r \overset{\text{R}}{\leftarrow} R$, and sends to the receiver $c \leftarrow E(\text{pk}, m; r)$, where E is some public-key encryption system. Then, to open, S sends r, and the receiver verifies that $c = E(\text{pk}, m; r)$.

  Assuming that E is semantically secure, then this scheme is computationally hiding. But it's not binding: public-key encryption may not be committing: there could very well exist (and there do, for some choices of E) (m_0, r_0) and (m_1, r_1) such that $E(\text{pk}, m_0; r_0) = E(\text{pk}, m_1; r_1)$. (Here, m is the plaintext and r is the randomness).

  If this is a bit complicated, relax it to a symmetric encryption scheme, in which case S opens the commitment by revealing K, and R verifies by checking that $c = E(k, m)$. This is once again computationally hiding, but it's not binding: even the one-time pad gives S multiple options for opening the commitment.

  The point is, encryption in general is not committing: to get binding, it's necessary to have an additional constraint of collision-resistance.

Let's go on to a more complicated commitment scheme; there is a classic result that, given a generator, one can obtain a commitment scheme. This construction will be computationally hiding, but perfectly binding. This is a little more general because it depends only on a PRG (since one-way permutations or one-way functions imply PRGs; thus, one can obtain commitments from one-way functions).

Let $G : \{0,1\}^n \to \{0,1\}^{3n}$ be a PRG. Then, R chooses a random seed $s \overset{\text{R}}{\leftarrow} \{0,1\}^{3n}$ and sends it to S. Then, the sender chooses $s \overset{\text{R}}{\leftarrow} \{0,1\}^n$. Then, it sends to R the commitment

$$\text{commit}(b) = c(s, b, r) = \begin{cases} G(s), & \text{if } b = 0 \\ G(s) \oplus r, & \text{if } b = 1. \end{cases} \tag{1}$$

Then, to open the commitment, S sends (b, s) to R, which verifies that (1) is true.

To show that this is computationally hiding, suppose R is the adversary. Then, because G is secure,

$$\left\{s \overset{\text{R}}{\leftarrow} \{0,1\}^n, r \overset{\text{R}}{\leftarrow} \{0,1\}^{3n} : (G(s), r)\right\} \overset{\text{c}}{\approx} \left\{R \overset{\text{R}}{\leftarrow} \{0,1\}^{3n}, r \overset{\text{R}}{\leftarrow} \{0,1\}^{3n} : (R, r)\right\}$$
$$\overset{\text{c}}{\approx} \left\{s \leftarrow \{0,1\}^n, r \overset{\text{R}}{\leftarrow} \{0,1\}^{3n} : (G(s) \oplus r, r)\right\}.$$

This is because a distinguisher for the latter two distributions can be used to break security for G, so this scheme is computationally hiding. Notice that thus far, $3n$ isn't necessary: you could just use n; it comes up in binding.

This construction is actually perfectly binding: to win the binding game, the adversary need to find two seeds $s_0, s_1 \in \{0,1\}^n$ such that $c(s_0, 0, r) = c(s_1, 1, r)$. However, this happens iff $G(s_0) = G(s_1) \oplus r$, which only happens if $G(s_0) \oplus G(s_1) = r$. But r is random in $\{0,1\}^{3n}$, but there are 2^n possible values for s_0 and s_1, i.e. 2^{2n} possible values on the left, and 2^{3n} possible values on the right. Thus, given a random r, it is extremely unlikely that there exist s_0 and s_1 such that S can win the binding game; specifically, the adversary's advantage, the probability that r has this property, is $1/2^n$, which is negligible. This is true no matter the computational power of A, so this scheme is perfectly binding.

5. Commitments II: 4/17/14

"You can earn a lot of money in consulting simply by telling people that symmetric encryption is not binding."

Lecture on Tuesday was rescheduled/cancelled because the CSO of Yahoo gave a talk at about the same time instead.

Recall that a commitment scheme is a scheme in which Alice has a bit b, and sends commit(b) to Bob. Later, she opens it by sending more information to Bob; a scheme is hiding if Bob cannot detect b from commit(b) before the scheme is opened, and is binding (both of these were given some computational assumptions on Alice and Bob) if Alice cannot open a scheme in two different ways with more than non-negligible probability

We saw a few computationally hiding schemes: if $f : X \to Y$ is a one-way permutation or collision-resistant hash function and $h : X \to \{0,1\}^\ell$ is hardcore for f, then commit$(b \in \{0,1\}^\ell)$ is given by choosing $x \overset{R}{\leftarrow} X$ and then sending $(f(x), h(x) \oplus b)$, and to open it, send (b, x).

Here's a bad commitment scheme: choose a random $k \in \mathcal{K}$, fixed for all messages; then, commit$(b \in \{0,1\}^{128})$ outputs AES(k, b). This is not good because it gives the same value if the same b is committed twice, so the adversary gains information, so it's not hiding. If one chooses a new key k for each message, then this is certainly computationally hiding, but this commitment can be opened as some different b' by choosing an arbitrary $k' \in \mathcal{K}$ and setting $b' \leftarrow \text{AES}^{-1}(k', c)$, so (b', k') is a valid opening. This is a rookie mistake; the scheme is not binding! The mistake is that symmetric encryption is in general not binding.

We also saw a perfecty binding and computationally hiding commitment scheme that builds from a PRG $S \to S^3$. Using something called the Hill construction, PRGs can be made from one-way functions, though this is so ugly that it would never be used in practice; it's useful mainly as a conceptual tool.

There are also perfectly hiding commitments, which are thus computationally binding. For example, given a one-way permutation $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$, called NOVY (with, apparently, a very cute proof of security). One-way permutations aren't exactly natural objects, unlike one-way functions, which are much easier to construct and don't require algebra (e.g. AES, SHA-256). Another construction begins with a function $f : \{0,1\}^{2n} \to \{0,1\}^n$ that is collision-resistant. Finally, the Peterson commitment scheme is based on the discrete logarithm: suppose $|G| = p$ and g and h are generators of G. Then, commit$(b \in \mathbb{Z}_p^*)$ picks $r \overset{R}{\leftarrow} \mathbb{Z}_p^*$, and sends $c = g^b h^r \in G$. This is perfectly hiding because r is random, so g^r is a random element in G, and it's computationally binding (open it by sending b and r, and then verifying), because $H : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to G$ given by $H(x, y) = g^x h^y$ is collision-resistant, assuming $\text{Dlog}_g(h)$ is hard. (Here, it helps that p is prime.)

Another algebraic commitment scheme is based on RSA. Let $N = pq$ and choose e such that $\gcd(e, \varphi(N)) = 1$. For $g \in \mathbb{Z}_N^*$ a generator, commit$(b \in \{0, 1, \dots, e-1\})$ chooses an $r \overset{R}{\leftarrow} \mathbb{Z}_N^*$ and sends $c = g^b r^e \pmod N$. This is perfectly hiding, because g^r is random garbage. Then, it's opened in the usual way (send enough information to verify); this is computationally binding becayse $H : \{0, \dots, e-1\} \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ given by $H(b, r) = g^b r^e \pmod N$ is collision-resistant assuming $\sqrt[e]{g} \pmod N$. But this is just the RSA problem, so it's probably hard.

Notice that these last two were actually variations on the same theme; most commitment schemes are supposed to be fast (and thus symmetric), but these last two have a neat porperty: that they're homomorphic. That means that, given commit(b_1) and commit(b_2), one can compute commit$(b_1 + b_2)$ (e.g. in the Peterson scheme, $(g^{b_1} h^{r_1}, g^{b_2} h^{r_2}) \mapsto g^{b_1+b_2} h^{r_1+r_2+r'}$ for some $r' \overset{R}{\leftarrow} R$, and RSA is similar). Until very recently, it was an important open problem as to whther fully homomorphic encryption exists (i.e. computing on any computable function f); of course, it's much slower than any of the schemes discussed above. But this has cool theory, e.g. commiting a search string, running Google's search algorithm on it, and thus committing the result, even though the state of the art is abysmally slow and doesn't make this work. Another useful application of this is if one has a proof-checking algorithm, it can be used for zero-knowledge proofs (e.g. the algorithm operates on the committed data, in some sense checking if the proof is correct without actually seeing it).

Let's take a closer look at the perfectly hiding scheme from a collision-resistant hash function (e.g. SHA-256) $H : \{0,1\}^T \to \{0,1\}^t$. Given some $b \in \{0,1\}^T$, choose some $r \overset{R}{\leftarrow} \{0,1\}^T$, and send $[H(r), r \oplus b]$. This *seems* perfectly hiding, because there are lots and lots of different r' that hash to the same value, but there's no guarantee that H doesn't leak bits of r: maybe all of the preimages of $H(r)$ start with a 0, in which case the first bit of b is leaked. Thus, this is *not* a good scheme!

The problem is to extract something uniform from the hash function; in other words, given c, r isn't uniformly random in $\{0,1\}^T$. The technique of turning nonuniform randnomness into uniform randomness uses a very general technique called an extractor.

Definition. A family $\mathcal{H}$ of hash functions $X \to Y$, where $|Y| < |X|$, is an ε-universal hash function (ε-UHF) for an $\varepsilon \in [0,1]$ if for all distinct $x, x' \in X$,

$$\Pr\left[h \overset{\mathrm{R}}{\leftarrow} \mathcal{H} : h(x) = h(x')\right] < \varepsilon.$$

This idea was introduced in the theory of data structures (for hash tables), and co-opted into crypto, where it seems the most use these days. The collision-resistance property is very different (the attacker doesn't know what the hash function is), which makes it possible to explicitly construct universal hash functions information-theoretically; they're much simpler and definitely not cryptographic hash functions.

Note that $\varepsilon \geq 1/|Y| - 1/|X|$, and we can actually achieve this lower bound.

It turns out that since it takes more time to insert into a hash table when there's a collision... so there's a timing attack (of course) that allows one to recover the original hash function being used. These are actually some of the oldest timing attacks, from the early 90s or so.

Example 5.1. Let p be prime and $X = (\mathbb{Z}_p)^n$ and $Y = \mathbb{Z}_p$. For any $t \in \mathbb{Z}_p$, let

$$h_t(m_0, \ldots, m_{n-1}) = \sum_{j=0}^{n-1} m_j t^j,$$

and let $\mathcal{H} = \{h_t \mid t \in \mathbb{Z}_p\}$, giving us a family of p hash functions that are easy to evaluate.

Claim. The $\mathcal{H}$ described above is $(n-1)/p$-UHF, i.e. if one chooses a random t, the probability that one obtains a collision is negligible.

Proof. The magic word is "roots:" if $\overline{m} \neq \overline{m}'$ and $h_t(\overline{m}) = h_t(\overline{m}')$, then the two polynomials evaluate to the same thing at t:

$$\sum_{i=0}^{n-1} m_i t^i = \sum_{i=1}^{n-1} m_i t^i.$$

But two polynomials of degree $n-1$ can agree on at most $n-1$ points (since their difference is a nonzero polynomial of degree at most $n-1$, so it has at most $n-1$ roots). Thus, the probability that a collision exists is at most $(n-1)/p$ (number of roots over the number of possible choices for t). $\boxtimes$

Intuitively, $p \gg n$, so the constant factor $(n-1)/p$ is small.

This is a very classic example of UHFs, especially when $n = 2$ (so everything is linear).

Before returning to commitment schemes, let's diverge again into measures of randomness; there is more than one, and this is significant. Suppose R is a random variable on a set Ω of size $|\Omega| = n$. Let U be the uniform distribution on Ω.

Definition.

- We say that R is δ-uniform if

  $$\delta = \Delta(R; U) = \frac{1}{2} \sum_{x \in \Omega} \left| \Pr[R = x] - \frac{1}{n} \right|.$$

  This just is the statistical distance from R to U, which makes sense, but isn't the optimal measure for crypto.
- The guessing probability is

  $$\gamma(R) = \max_{x \in \Omega} \{\Pr[R = x]\}.$$

  This is a number between 0 and 1. Intuitively, the lower the guessing probability, the more closely a distribution resembles a uniform distribution when one guesses.
- The collision probability is

  $$\kappa(R) = \sum_{x \in \Omega} \Pr[R = x]^2.$$

  This asks how likely it is that one gets a particular x twice (corresponding to a collision). Notice also that $\gamma(U) = \kappa(U) = 1/n$.
- $-\log_2 \gamma(R)$ is called the min-entropy of R, and $-\log_2 \kappa(R)$ is called the Rényi entropy.

Notice that $\kappa(R)$ is the ℓ^2 norm of the distribution vector, and $\gamma(R)$ is the ℓ^∞ norm! This means that for any exponent, there's a different measure of randomness (e.g. for ℓ^3, the tri-collision probability).

It's a fact that if R is δ-uniform on Ω and $|\Omega| = n$, then

$$\frac{1+4\delta^2}{b} \leq \kappa(R) \leq \gamma(R) \leq \frac{1}{n} + \delta.$$

This follows from statements such that $\ell^2(R) \leq \ell^1(R)$, and so on.

Now, back to crypto.

Lemma 5.2 (Leftover Hash Lemma)**.** *Let $\mathcal{H} = \{h : X \to Y\}$ be a ε-UHF family of hash functions, and let R be a random variable over X, and choose $h \overset{\text{R}}{\leftarrow} \mathcal{H}$ be chosen* independently *of R. Then, $(h, h(R))$ is δ-uniform over $\mathcal{H} \times Y$, where*

$$\delta < \frac{1}{2}\sqrt{|Y| \cdot (\kappa(R) + \varepsilon) - 1}.$$

This specific calculation for δ isn't so useful, but the fact that this distribution is close to uniform, even if the chosen h is public, is very useful for cosntructing extractors. Effectively, as soon as R has some minimum emtropy, then we can get a close-to-uniform value out. This is helpful for, for example, Diffie-Hellman key exchange, which allows this to convert a uniform group element into a nearly uniform bit string that can be used as an AES key.

Example 5.3. Suppose R is uniform on $S \subseteq \Omega$, where $|s| = 2^{300}$, so that $\kappa(R) = 1/|S| = 2^{-300}$. Suppose that $\mathcal{H}$ is a collection of functions $\Omega \to Y$ that is a 2^{-100}-UHF, and that $|Y| = 2^{100}$. Then, Lemma 5.2 tells us that $(h, h(R))$ is δ-uniform for some δ after fiddling with the parameters.

It's easy to make $\mathcal{H}$ as universal as you want; just pick larger and larger primes.

Finally, let's return to the commitments that we were committed to covering. Let $H : \{0,1\}^T \to \{0,1\}^t$ be a collision-resistant hash function; then, S choses an $r \overset{\text{R}}{\leftarrow} \{0,1\}^T$ and an $h \overset{\text{R}}{\leftarrow} \mathcal{H}$, and let $c_1 \leftarrow H(r)$ and $c_2 \leftarrow h(t) \oplus b$ (which is the extracted entropy). Then, let commit$(b) = (h, c_2)$. This is computed extremely quickly; and then, to open it, send (r, b) as before. Then, this is perfectly hiding (by Lemma 5.2) and computationally binding (since H is collision-resistant), but it has no homomorphic properties. This won't be the last application of the leftover hash lemma.

6. Oblivious Transfer: 4/22/14

"If you ask me how hard it is, I'll tell you: I think everything is easy. But I don't even know where to start."

The setup for oblivious transfer is a sender S that has messages $m_1, \dots, m_n$ and the receiver R receives exactly one m_i, but S doesn't know the value of i. This is a somewhat strange primitive, but ends up being a useful building block in a diverse array of cryptographic protocols. Moreover, it's a crucial ingredient in something known as Strong Private Information Retrieval, e.g. where a database can confirm that Alice has accessed exactly one element, but doesn't know which. There's another protocol called adaptive oblivious transfer which is a slight generalization, but nonetheless useful.

The notion of security for an oblivious transfer is called half simulation. There are two types: security from the sender and security from the receiver.

Definition.

(1) For security for the receiver, the goal is to hide i from a malicious sender. Thus, View$_S(i, (m_1, \dots, m_n))$ denotes what the sender sees when $s \leftarrow m_1, \dots, m_n$ and i is chosen at random. Thus, the view is a random variable, and the system is considered to be secure when for all all $i \neq j \in \{1, \dots, n\}$ and all $m_1, \dots, m_n \in \mathcal{M}$

$$\text{View}_S(i, (m_1, \dots, m_n)) \overset{\text{c}}{\approx} \text{View}_S(j, (m_1, \dots, m_n)).$$

(2) For security of a sender against a malicious receiver, it's a little trickier: how does one formalize that R doesn't learn anything other than m_i? This goes as follows: for all $m_1, \dots, m_n \in \mathcal{M}$ and all efficient adversaries $\mathcal{A}$, there exists an efficient simulation $\mathcal{S}$ that only receives m_i such that the output of $\mathcal{A}$ interacting with S is computationally indistinguishable from that of $\mathcal{S}$.

$$\begin{tikzcd}
& m_1, \dots, m_n \arrow[d] \\
\boxed{\mathcal{A}} \arrow[r, shift left=2] \arrow[r, shift right=2] \arrow[d] & \boxed{S} \arrow[l] \\
\text{out}_1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
& m_1, \dots, m_n \arrow[d] \\
\boxed{\mathcal{S}} \arrow[r, shift left, "i"] \arrow[d] & \boxed{S} \arrow[l, shift left, "m_i"] \\
\text{out}_2 &
\end{tikzcd}$$

Then, we require that $\text{out}_1 \overset{c}{\approx} \text{out}_2$.

This is called half simulation because the simulation security definition is used for one side of the exchange, but not the other.

It's an easy exercise that oblivious transfer implies key exchange secure against eavesdropping (and it also implies a lot more for multi-party computation). But this means that it can't be built from AES-like techniques, and instead we need algebra. A simple construction uses something called blind signatures: given some key-generating algorithm $(sx, px) \leftarrow \texttt{KeyGen()}$, the sender S has a secret key sk and a public key pk, such that (after perhaps some more communications), the receiver R signs a message $m \in \mathcal{M}$, obtaining σ. Then, it's required for the signature to be unique, i.e. this σ is the only such one such that $V(\text{pk}, m, \sigma) = 1$. (Such schemes have been constructed, depending on factoring being hard, etc.)

This doesn't really work with existential unforgeability as we have seen it, so it has to be relaxed: given a public key pk, an adversary can make q signature requests for messages m, and this adversary wins if it can output $q+1$ valid message-signature pairs. A system is called unforgeable if it is secure against such games (in the usual probabilistic sense); this is the notion of sender security.

For recipient security, the adversary sends the challenger the public key pk and two messages m_0 and m_1, which chooses some $b \in \{0, 1\}$ and signs m_b. Then, the adversary tries to guess the value of b; the scheme is secure if the adversary's advantage ie negligible.

Here are some examples of this.

Example 6.1.

(1) From RSA, let KeyGen choose primes p and q and $N \leftarrow pq$, and lets 3 be the public exponent (so that $\gcd(3, \varphi(N)) = 1$). Then, the public key is (N, e), and the secret key is $(N, 3^{-1} \bmod \varphi(N) = d)$. We also want to have a hash function $H : \mathcal{M} \to \mathbb{Z}_N$ lying around. Thus, to turn this into a blind signature scheme, the receiver chooses an $r \overset{R}{\leftarrow} \mathbb{Z}_N$, and then send to S $y \leftarrow H(m) \cdot r^3 \in \mathbb{Z}_N$, so it can compute the cube root $z \leftarrow y^{1/3}$ in $\mathbb{Z}_N$. Then, the signature is $\sigma = z/r \in \mathbb{Z}_N$.

This ends up being consistent, because $\sigma^3 = (y^{1/3}/r)^3 = y/r^3 = H(m)$. But is it secure? For sender security, this boils down to the RSA assumption, assuming H is a random oracle. For recipient security, it seems reasonable, but the adversary can choose the public key; specifically, the adversary may choose N such that $3 \mid \varphi(N)$. This is hard to test (especially compared to whether $2 \mid \varphi(N)$, which boils down to p and q being odd) — or rather, it seems like it ought to be easy, but lots of work has been spent on it. Nonetheless, there are plenty of protcols created and used based on the assumption that determining whether $e \mid \varphi(N)$ is hard (for $e \geq 3$).

In any case, if $3 \mid \varphi(N)$, then the cubes are a strict subgroup of $\mathbb{Z}_N$, and in particular, r^3 isn't uniform in $\mathbb{Z}_N$. Thus, we might have that $H(m)$ doesn't have a cube root, in which case S learns this when it fails to compute z. But this means S learns something about the message signed, which is no good (specifically, whether it's a cubic residue or not).

For reference, this is a forty-year-old security protocol, but was only recently shown to be insecure; this is one of the important points of choosing the right definition!

The way around this is to make the public key contain a zero-knowledge proof that $3 \nmid \varphi(N)$ (which makes this protocol, and the oblivious transfer protocol based on that, secure).

(2) There's another protocol named BLS, which involves dealing with the notion of bilinear maps.

Definition. Let G be a group of prime order and g be a generator for G, and let G_T be some other group (called the target group). Then, a bilinear map on G is a map $e : G \times G \to G_T$ such that:

- e is efficiently computable,
- $e(g, g) \neq 1$, and
- for all a, b, $e(g^a, g^b) = e(g, g)^{ab}$.[12]

This lends itself nicely to a signature scheme: let p be prime and g generate G, a group of order p. Then, let $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p$, and the secret key is α and the public key is g^α. Then, letting $H : \mathcal{M} \to G$ be a hash as before, the signature is $S(\text{sk}, m) = H(m)^\alpha$, and verification involves checking that $e(g^\alpha, H(m)) = e(g, \sigma)$ (since $e(g^a, h) = e(g, h^a)$).

Theorem 6.2. *The BLS signature is existentially unforgeable if H is a random oracle and the computational Diffie-Hellman assumption holds in G.*

[12]Interestingly enough, these were discovered by a mathematician who was in jail for being a deserter from the French army in World War II (though he called himself a conscientious objector). This period of his life was so productive he later wrote a letter suggesting that all mathematicians be required to spend two years in jail!

This is all fine, but the point is to turn this into a blindable scheme as follows: the receiver has access to the public key and some $m \in \mathcal{M}$. Then, it sends $y \leftarrow H(m) \cdot g^r \in G$, and the sender responds with $z \leftarrow y^\alpha$. The signature thus is $\sigma \leftarrow z/(g^\alpha)^r$. Now, no zero-knowledge proof is needed to validate the public key; that g is a generator is as easy as checking $g \neq 1$ (since p is prime). It's also a unique signature.

Blind signature schemes are used for anonymous voting systems, anonymous cash, and so on; today, we'll be using them to construct oblivious transfer.

We start with unique blind signatures $\mathcal{M}_{\text{sig}}$ parameterized by $[n] = \{1, \dots, n\}$ and a hash $H : [n] \times \mathcal{M}_{\text{sig}} \to \{0,1\}^\ell$ that is a random oracle. Then, after a public key and secret key are chosen, the sender generates n ciphertexts via $\sigma_i \leftarrow S(\text{sk}, i)$ (where i is in a sense the message sent by the receiver), and then the final ciphertext is $c_i \leftarrow H(i, \sigma_i) \oplus m_i$: in a sense, we use the hash as a one-time pad to enode m_i. Then, the sender sends $\text{pk}, c_1, \dots, c_n$ to the receiver, which outputs $c_i \oplus H(i, \sigma_i)$.

This will end up being secure, though this has to be proven. But there's a very nifty property: this protocol of asking for messages repeatedly can be repeated over and over. For example, a Netflix user can request a movie, have a one-time penalty of downloading the entire encrypted database, and then each purchase is a quick interaction with the server.

Recipient security follows from that for the blind signature, but sender security is a little harder. Intuitively, given an efficient adversary A, a trusted T generates a public key and a secret key (pk, sk) via KeyGen, and creates $c_1, \dots, c_n \overset{R}{\leftarrow} \{0,1\}^\ell$ (encryption of garbage), and sends the public key and these $c_1, \dots, c_n$ to the adversary. Then, the adversary sends back some blind signature and a random oracle query for $H(j, \sigma)$. If the verification of the signature indicates it's correct, then T indicates m_j, and we send $c_j \oplus m_j$ to the adversary. But if verificaton fails, then we return random junk c_j. Then, the adversary may output.

The point is, we are able to program the random oracle, and the system is secure if we can produce the same thing the adversary does in some sense. But we're toast if the adversary can somehow issue two queries with the same valid signature, since we can't query the trusted source T, so if this is built on a forgeable scheme, then this can't work, and in fact this simulation can be used to forge signatures if such an efficient adversary exists.

7. Oblivious Transfer II: 4/24/14

"I don't understand; a filesystem is just a sequence of bits. Just encrypt it — what more is there to say?"

Recall that last time, we introduced oblivious transfer, where a sender S has $x_1, \dots, x_n \in X$ and the receiver R requests some x_i for $i \in [n]$; the system has sender security if R can learn x_i and nothing else, and has receiver security if S learns nothing abut i. The example we saw was based on blind signatures, and is a very natural protocol: it can be used to build a large number of other cryptogaphic protocols.

For example, here's a protocol due to Bellare and Micali, which is based on the computational Diffie-Hellman assumption. Let G be a group of prime order p with a generator $g \in G$. This will be a 1-out-of-2 oblivious transfer, i.e. the sender has two values $m_0, m_1 \in \{0,1\}^\ell$, and the receiver has a bit $b \in \{0,1\}$ (corresponding to whether it asks for m_0 or m_1). We also need a hash function $H : G \to \{0,1\}^\ell$. Then, here's how the protocol works:

(1) First, the sender gets a $c \overset{R}{\leftarrow} G$ and sends it to the receiver.
(2) Then, the receiver gets a $k \overset{R}{\leftarrow} \mathbb{Z}_p$ and generates two ElGamal secret keys $y_b \leftarrow g^k$ and $y_{1-b} \overset{R}{\leftarrow} c/g^k$¡ so that R knows the discrete log of one of y_0 or y_1, but not both (intuitively, this would allow R to compute the discrete log of c). Then, R sends y_0 and y_1 to S.
(3) The sender checks that $y_0 \cdot y_1 = c$, and aborts if not (since this would destroy sender security).
(4) Now, the sender computes a standard ElGamal encryption: it chooses $r_0, r_1 \leftarrow \mathbb{Z}_p^*$ and sets $c_0 \leftarrow (g^{r_0}, H(y_0^{r_0}) \oplus m_0)$ and $c_1 \leftarrow (g^{r_1}, H(y_1^{r_1}) \oplus m_1)$ and sends them to the receiver.
(5) Finally, the receiver decrypts $c_b = (v_0, v_1)$ (standard ElGamal decryption) and outputs $H(v_0^k) \oplus v_1 = m_b$.

For recipient security, we have information-theoretic security: the sender sees the same distribution on y_0 and y_1, no matter the value of b, so it can't tell the value of b from them. Sender security is a little more involved, but is similar to a proof we did last time. depending on the computational Diffie-Hellman assumption.

There is a similarly elegant but slightly more complicated protocol in which the sender has information-theoretic security and the receiver has security based on the Diffie-Hellman assumption.[13] However, neither this nor the previous one are efficient enough for the real world.

For this one turns to oblivious transfer with preprocessing. Before any messages are received, the sender builds many pairs y_0, y_1), and the recipient has some bit k and y_k (from the same source). Then, the online step is for R to

[13]Naor, M and Pinkas, B. *Efficient oblivious transfer protocols.* SODA '01.

send $c \leftarrow b \oplus k$ to S, which responds with $z_0 = y_c \oplus x_0$ and $z_1 = y_{1-c} \oplus x_1$, and the receiver outputs $z_b \oplus y_k$. This is very fast: three $\oplus$ computations take nearly no time.[14] As for why $z_b \oplus y_k = y_b$, this can be seen by a brute-force checking of cases: if $b = 0$, then

$$z_b \oplus y_k = (y_c \oplus x_0) \oplus y_k = y_k \oplus x_0 \oplus y_k = x_0.$$

The case $b = 1$ is similar. Note also that c can be generated from a hash function starting with some publicly known value.

Recipient privacy is pretty self-evident: S only sees a one-time pad encryption of k, and sender privacy follows because R only knows y_0 or y_1, and the other has one-time pad security.

This leads to a notion called oblivious polynomial evaluation, in which the sender S has some $f \in \mathbb{F}_p[x]$ and the receiver has some $y \in \mathbb{F}_p$; then, at the end of the protocol, R has $f(y)$; the goal is for the amount of communication to be $o(\deg f)$. This is useful because polynomial oblivious transfer implies 1-out-of-n oblivious transfer: if R has some $i \in [n]$ and S has $a_1, \dots, a_n$, so it picks a random f of degree $n + 1$ such that $f(i) = a_i$ for all $i \in [n]$ (so that the value of f at i doesn't leak any information about the other values — but this only admits one query), and thus can overlay this on top of an existing oblivious transfer protocol.

It also implies a comparison protocol, where S has a $b \in \mathbb{F}_p$ and R has an $a \in \mathbb{F}_p$; then, they want to check if they're equal, but not leak any more information if they're not. In this case, S picks some $r \xleftarrow{R} \mathbb{F}_p^*$ and lets $f(x) = r(x - a)$; then, if $a = b$, then $f(b) = 0$, and otherwise, its value is random in $\mathbb{F}_p^*$. (The specifics of the transfer boil down to the polynomial evaluation protocol from before.) Often, for p large, one can use $\mathbb{F}_p$ instead of $\mathbb{F}_p^*$; their statistical dependence lessens and lessens. This protocol is useful in, for example, password authentication.

This can also be used to determine set membership! Here, the sender has a set $A = \{a_1, \dots, a_n\} \subseteq \mathbb{F}_p$, and the recipient has a $b \in \mathbb{F}_p$ and wants to know if $b \in A$. This might be useful if you want to check if you're on the no-fly list for the TSA without revealing your identity. This boils down to polynomial evaluation of

$$f(x) = r \cdot \prod_{i=1}^{n} (x - a_i)$$

at b, where r is random in$\mathbb{Z}_p^*$.

Finally, one can use this to make oblivious PRFs $F : \mathcal{K} \times X \to Y$. The idea is that the sender has a $k \in \mathcal{K}$ and the receiver has an $x \in X$, and wants to know $F(k, x)$ without leaking x to S or k to R. One technique is the Naor-Reingold PRF, which is efficient (and doesn't depend on the polynomial evaluation protocol described above). It leads to a nice protocol for set membership: first, S chooses a $k \leftarrow \mathcal{K}$, and then sends $F(k, a_1), \dots, F(k, a_n)$; then, the receiver can determine $F(k, b)$ without k or the sender knowing b, and then determine if the value is in one of the $F(k, a_i)$. (Then, this implies set intersection, etc.)

Private Information Retrieval (PIR). This is related to oblivious transfer; here, the sender has some database $\{x_1, \dots, x_n\} \subseteq \{0, 1\}^\ell$, and the receiver has an $i \in [n]$, and learns x_i and possibly other things, but the sender learns nothing from R (we only care about recipient privacy). Trivially, S could send everything, but the goal is to minimize communication. A classic example is a patent database: it's fine to see other patents than the one extracted, but one might not want to broadcast which patents they read.

For a warm-up, there's a more practical but much less cool protocol called two-server PIR, where S_1 holds $x_1, \dots, x_n \in \mathbb{F}_p$ and S_2 also has a copy of $x_1, \dots, x_n \in \mathbb{F}_p$, and S_1 and S_2 don't talk to each other. Then, the recipient talks to both S_1 and S_2 an obtains x_i and possibly other stuff, and as long as S_1 and S_2 don't communicate, we have privacy.[15]

There's a somewhat trivial, but not as trivial, protocol that takes $O(\sqrt{n})$ traffic.[16] Introduce the matrix

$$A = \begin{bmatrix} x_{1,1} & \cdots & x_{1,\sqrt{n}} \\ \vdots & \ddots & \vdots \\ x_{\sqrt{n},1} & \cdots & x_{\sqrt{n},\sqrt{n}} \end{bmatrix} \in \mathbb{F}_p^{\sqrt{n} \times \sqrt{n}}.$$

Here, the sequence of n elements is represented as a matrix, so the indexing is rearranged slightly (and R is interested in (i, j)). The naïve idea of asking S_1 for the column in question and S_2 for the row in question leaks information about which row or column the object of interest is in, which isn't good. But instead, one could obtain that from Ae_i. So to mask this, R chooses two vectors $v_1, v_2 \leftarrow \mathbb{F}_p^{\sqrt{n}}$ such that $v_1 + v_2 = e_i \in \mathbb{F}_p^{\sqrt{n}}$. Then, from the perspective of S_1 or S_2, these v_i are uniformly random. Then, they send back $Av_1 = u_1$ and $Av_2 = u_2$; then, the recipient adds them

[14] A similar protocol is done for RSA signature schemes (and can be generalized to others), where the time-intensive computations are done offline, making signing things much faster or on small devices such as smart cards or watches or whatnot.

[15] You can probably guess how to generalize this to n-server PIR.

[16] In general, when you see something that takes $O(\sqrt{n})$ time, space, or traffic, always think about matrices!

to get $u_1 + u_2 = A(v_1 + v_2) = Ae_i$, and so we get the point the recipient is interested in. Since v_1 and v_2 are both independent of i and j, then this is secure. It's fast from a network standpoint, but matrix multiplication requires the server to look at every database entry, which seems inefficient. But this is necessary, because if the server doesn't need to look at an element, then the user doesn't need it somehow (which is mitigated by the presence of the other server, but only by a factor of 2). Then, $2\sqrt{n}$ elements are sent back and forth, which is the correct amount.

It turns out there's a way to make 2-server PIR work with $O(\sqrt[3]{n})$ communication, and this is the best one can do.

Can we do better than that with 3 servers? It was an open problem for a long time, but in 2009 a better bound of $O(2^{6\sqrt{\log n \log\log n}})$ was discovered. This is $\ll n^\varepsilon$ for all ε, so it was quite a surprise. And this might not even be a tight bound. Importantly, all of these results are information-theoretic, which can be done because the servers don't collude.

However, returning to the single-server case, one can show that a sublinear PIR algorithm implies collision-resistance, and thus we'll need some complexity assumptions. However, for all $\varepsilon > 0$, there's an $o(n^\varepsilon)$ protocol using additively homomorphic encryption, and there's an $o(\log n)$ protocol based on an assumption called the ϕ-hiding assumption. (This assumption is still open, but the professor hopes it's false.)

Let $\ell_1, \dots, \ell_n$ be the first n primes, so that $\ell_n \approx n \ln n$. Suppose the sender S has a database $x_1, \dots, x_n \in \{0, 1\}$ (a database of bits, like a theoretician's view), and the receiver wants to know about some $i \in [n]$. Then:

(1) R chooses $p, q \xleftarrow{R} \text{Primes}(\lambda)$ (i.e. λ-bit functions), for some λ, and such that $p \equiv 1 \pmod{\ell_i}$. Then, it sets $N \leftarrow pq$, and chooses a $w \xleftarrow{R} \mathbb{Z}_N$ such that w doesn't have an ℓ_i^{th} root in $\mathbb{Z}_N$ (i.e. that $w^{(p-1)/\ell_i} \not\equiv 1 \pmod{p}$). Then, send N and w to the server.

(2) S starts taking powers of w, getting
$$a \leftarrow w^{\prod_{j=1}^n \ell_j^{x_j}} \in \mathbb{Z}_N,$$
in some sense taking exactly the primes such that $x_n = 1$. This is a pretty crazy exponentiation, but S sends a to R, so there's not much traffic.

(3) Finally, the receiver calculates $a^{(p-1)/\ell_i} \bmod p$, which is 1 iff $x_i = 1$, and otherwise, $x_i = 0$.

Why does this work?
$$a^{(p-1)/\ell_i} = \left(w^{\prod_{j=1}^n \ell_j^{x_j}}\right)^{p-1} \equiv 1 \text{ iff } x_i = 1,$$
The communication is 3 elements of $\mathbb{Z}_N$, but $N = o(n \ln n)$, so there are $o(\log n)$ communication bits (which is very low). As for security, the sender sees n and w. Do they leak information about i? Unfortunately, this involves the standard trick in crypto: that they don't.

More formally (and elegantly), the ϕ-hiding assumption claims that the following two distributions are indistinguishable for all $\ell < \lambda^{o(1)}$: choose $\{p, q \xleftarrow{R} \text{Primes}(\lambda)\}$ such that $p \equiv 1 \bmod \ell$, and $s \xleftarrow{R} \mathbb{Z}_N$ such that s isn't an ℓ^{th} residue; then, output (pq, s), compared with the distribution where ℓ might or might not divide $\phi(n)$ (unlike the previous distribution, where $\ell \mid \phi(n)$), implying the name. This seems suspicious, but is still open.

8. Zero-Knowledge Proofs: 4/29/14

"Now, whenever you think about graph isomorphism, you're going to think about milk."

Before discussing interactive proof systems, let's mention the complexity heirarchy: there are deterministic polynomial-time functions **P**, and on top of that **NP** and co**NP** both containing **P**, followed by a whole polynomial heirarchy, and PSPACE. There's more on both ends, but this is what we want for now.

It's useful to have that if $L \in \mathbf{NP}$, then there exists a deterministic polynomial-time algorithm $V : \Sigma^* \times \Sigma^* \to \{0, 1\}$ such that $x \in L$ iff there exists a $p \in \Sigma^*$ such that $V(x, p) = 1$. V is said to be a verifier, and p is a short proof that $x \in L$.

We can extend this model in a number of ways.

(1) What if V is randomized? Specifically, we require that if $x \in L$, $\Pr[V(x, p) = 1] \geq 2/3$ for some $p \in \Sigma^*$. This complexity class is called MA, for Merlin-Arthur (thanks to a scenario where Arthur calls upon his oracle Merlin to answer questions).

(2) What if V can only read the first 3 bits of **P**? This class is known as PCP, and one of the deepest theorems in computer science is that PCP = **NP**, which could take a whole quarter to prove.

(3) What if the prover and verifier are allowed to interact? Here, P and V both share x, but are allowed to share messages back and forth, and eventually V emits one of 0 or 1. The class of strings which P can convince V are in the language is known as IP (interactive proofs).

The last class, IP, is the most useful for crypto, and will be focused upon here.

Definition. An interactive proof system for an $L \subseteq \{0,1\}^*$ is a two-party protocol[17] between a probabilistic polynomial-time verifier V and an infinitely powerful prover P sharing an $x \in L$ satisfying the following properties.

(1) Completeness: for all $x \in L$, V always accepts x after interacting with P.
(2) Soundness: for all $x \notin L$ and all "bad" provers P^*, V rejects with probability greater than[18] 1/2 after interacting with P^*.

Then, one can formally define IP to be all languages $L \subseteq \{0,1\}^*$ that are recognizable by an interactive proof system (i.e., there exist V and P such that the language recognized by them is L). Additionally, define IP(m) to be the set of languages in $\{0,1\}^*$ which are recognizable by an interactive proof of at most m rounds (P and V can send at most m messages together).

First, note that **NP** $\subseteq$ MA, since any deterministic verifier can be superseded by a randomized verifier. But, more interestingly, MA $\subseteq$ IP(1), given by the algorithm where P sends a proof, and V checks it. And then it seems that IP(1) $\subseteq$ IP(2) $\subseteq$ IP(3) $\subseteq \cdots$, but it turns out that MA = IP(m)!

However, if one forces V to be deterministic, then the class of languags recognized is exactly **NP**, so the fact that V can be randomized is what makes this very interesting: this is because it reduces to P sending one message to V, but then this is just polynomial-time verification.

Example 8.1. Let's look at the problem of graph isomorphism: if $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs, then they're considered isomorphic if there exists a $\pi : V_1 \to V_2$ such that $(v_1, v_2) \in E_1$ iff $(\pi(v_1), \pi(v_2)) \in E_2$ for all $v_1, v_2 \in V_1$.

The problem of determining whether two graphs are isomorphic is in **NP**, and therefore there's an interactive proof (where the prover just sends the isomorphism to the verifier; soundness and completeness follow). But graph non-isomorphism (i.e. proving two graphs aren't isomorphic) is also in IP, even though it's believed to not be in **NP**.[19]

If P and V are looking at two graphs (G_0, G_1), then V picks a $\pi \overset{\text{R}}{\leftarrow} S_n$ (i.e. the symmetric group on n elements, the set of permutations of $\{1, \ldots, n\}$), and a $b \overset{\text{R}}{\leftarrow} \{0,1\}$, and then sends to P the graph $G^* = G_b^\pi$. Then, the prover sends back a $b' \in \{0,1\}$ and the verifier accepts if $b = b'$. This could be called the "milk game," inspired by a similar testing experiment where the prover hid one glass of milk from the verifier and asked her to guess what it was.

In the case of an honest prover (completeness), it's infinitely powerful, so it can determine which of G_0 or G_1 was used, so it sends back that particular value of b, and it's sound, because a bad prover won't be able to come up with an isomorphism if there isn't one, so $\Pr[b = b'] = 1/2$.

Here's a few facts to keep in mind.

- The prover P only needs to be PSPACE, and sometimes, we'll want more efficient provers.
- There's another formalism called Arthur-Merlin games,[20] in which the verifier is only allowed to send random values to the prover, which then responds; in this case, the verifier is called Arthur, and and the prover is called Merlin. This class of problems is called AM*. Then, there's a theorem that AM* = IP. This is kind of interesting; if your questions are random enough, you don't actually need to think about what to ask.

So, where does IP actually live in the complexity heirarchy? There's a beautiful result:

Theorem 8.2. IP = PSPACE.

Proof sketch. We'll start by showing that co**NP** $\subseteq$ IP, which is already surprising. It's enough to do this for an **NP**-complete laguage, e.g. that of 3-colorable graphs. Let $G = (V, E)$ and $n = |V|$. Then, the goal is to prove that G is not 3-colorable.

Define a polynomial from G as

$$P_G(x_1, \ldots, x_n) = \prod_{(u,v) \in E} (x_u - x_v).$$

Here, $x_i \in \{0,1,2\}$ should denote the color of vertex i, so if $(x_1, \ldots, x_n)$ is an invalid coloring, then $P_G(x_1, \ldots, x_n) = 0$, and if $(x_1, \ldots, x_n)$ is valid, $P_G(x_1, \ldots, x_n)$ is nonzero. We want to make it exactly equal to 1, so let $f(x) =$

[17]This can be formally defined as a set of two Turing machines, each of which can write to tapes read by the other, and so on, but that's not the point.

[18]It's possible to change the bound without any difference, since repeating a randomized algorithm several times decreases the porbability of a false positive.

[19]Curiously enough, it's known to be not **NP**-complete, and generating hard instances for it is difficult; heuristics for it work for many kinds of graphs.

[20]This was invented by Babai of the University of Chicago in the 1990s, not by King Arthur (and Merlin) of Camelot in the Middle Ages.

$(5/4)x^2 - (1/4)x^4$, which was chosen so that $f(0) = 0$ and $f(1) = f(2) = f(-1) = f(-2) = 1$. Then, redefine

$$P_G(x_1, \ldots, x_n) = \prod_{(u,v)\in E} f(x_u - x_v).$$

Now, $P_G(x_1, \ldots, x_n) = 1$ iff $(x_1, \ldots, x_n)$ is valid, and is 0 otherwise. Now, define

$$S = \sum_{x_1\in\{0,1,2\}} \sum_{x_2\in\{0,1,2\}} \cdots \sum_{x_n\in\{0,1,2\}} P_G(x_1, \ldots, x_n),$$

which is the number of valid 3-colorings of G (so we want to know whether $S = 0$).[21].

Now, we introduce the prover and verifier, which have access to P_G and S. Choose a prime $p > 4\,|V|\,\lambda$ (where λ is some large security parameter depending on $|V|$), and define

$$P_1(x) = \sum_{x_2\in\{0,1,2\}} \sum_{x_3\in\{0,1,2\}} \cdots \sum_{x_n\in\{0,1,2\}} P_G(x, x_2, \ldots, x_n).$$

Then, $\deg(P_1) < 4n$, 4 from f. Then, P sends the coefficients of some polynomial $\widehat{P}_1$ to V (an honest prover would send over P_1), which checks that $\widehat{P}_1(0) + \widehat{P}_1(1) + \widehat{P}_2(2) = S$, and aborts if this isn't true, since this is true for P_1. Then, the verifier chooses an $r \xleftarrow{\text{R}} \mathbb{F}_p$ and emits 1 if $P_1(r) = \widehat{P}_1(r)$ and 0 otherwise (since random polynomials are unlikely to have the same values on large $\mathbb{F}_p$).

But how does the verifier compute $S_1 = P_1(r)$? Well now, the verifier has $\widehat{P}_1(r)$ and the prover and verifier are trying to compute on the value

$$S_1 = \sum_{x_2\in\{0,1,2\}} \sum_{x_3\in\{0,1,2\}} \cdots \sum_{x_n\in\{0,1,2\}} P_G(x_1, \ldots, x_n),$$

so it's possible for the protocol to just recurse until the verifier can just compute the last two or three summations by brute-force. Then, it unravels everything, and then verifier is convinced that everything is correct.

Thus, this is a $2n$-round Arthur-Merlin protocol, and it's crucial that the randomness is chosen after the prover sends over its answer each time.

The general proof for PSPACE has very similar principles. ⊠

This also shows that $\text{AM}^* = \text{IP}$, too.

The idea is that for sufficiently large polynomials, if they agree at a random point, they almost certainly agree. This is an easy way to test whether two polynomials are equal, and comes up again and again in computer science. For example, one might test equality of sets by generating two polynomials whose roots are exatly the elements in each set, and then checking if the polynomials are equal. Note that this algebraic proof is the only one the professor knows about.

Interactive proof systems are said to be zero-knowledge if the proof doesn't leak any information to V that it doesn't already have.

Definition. A perfect zero-knowledge proof for a language $L \subseteq \{0,1\}^*$ is a proof system (P, V) for L (i.e. it is complete and sound) such that for every probabilistic polynomial-time verifier V^*, there exists a probabilistic polynomial-time simulator S^* such that for all $x \in L$, the following distributions are the same:

$$\{\text{transcript}(P(x) \longleftrightarrow V^*(x))\} = \{S^*(x)\}.$$

This means that anything that V^* could have learned from P can be determined from x already, so V^* doesn't learn anything from P, beyond the fact that $x \in L$. Note that if $x \notin L$, then there's no restriction; intuitively, if you're trying to prove the false fact that $x \in L$, then something is already going wrong, and information leaking is less of a concern.

The original paper by Goldwasser and Micali is a classic in crypto; yet it was rejected three times (albeit from the same journal) before being accepted. We'll write out an example now, but prove that it's zero-knowledge next time.

Remark. If perfectly hiding commitments exist, then every $L \in$ PSPACE (i.e. anything that can be proven interactively) has a zero-knowledge proof.

This was quite a surprise when it was discovered, though the proof boils down to showing that **NP** has zero-knowledge proofs (e.g. prove that 3-coloring has zero-knowledge proofs, which is easy), and then boost to PSPACE, which also isn't hard.

[21] This comes from a class of problems known as #**P** (called sharp-**P**), which asks for counting problems, where there's an integer answer, rather than just $\{0, 1\}$

Example 8.3. The example of quadratic reciprocity is very useful in crypto. Fix $N = pq$ and $x \in \mathbb{Z}_N^*$; the goal is to prove that x has a square root in $\mathbb{Z}_N^*$ (which is only true for one-quarter of the elements). Detecting this from the ground up is hard (albeit not as hard as factoring), and there's an easy interactive proof, given by just sending a square root over. But this isn't zero-knowledge.

In a zero-knowledge protocol, the prover is defending itself from a malicious verifier, so it'll use a blinding; it chooses some $r \overset{\text{R}}{\leftarrow} \mathbb{Z}_N$ and send over $a = r^2 \pmod N$. Then, the verifier chooses a $b \overset{\text{R}}{\leftarrow} \{0, 1\}$, and sends it to P. Then, the prover sends back $c \leftarrow r \cdot \alpha^b \in \mathbb{Z}_N$, where $\alpha \equiv x^2 \pmod N$, and the verifier checks that $c^2 = ax^b \pmod N$, and accepts if so.

What's going on is that the prover either sends r, or $r\alpha$; in the first case, where $b = 0$, the verifier learns the square root of a random square, which isn't of much use to it. But the blinding in the case $b = 1$ will make this zero-knowledge. The fact that b is randomly chosen is there to keep P honest: otherwise, if it were always 1, then P could cook up a seed such that every number is "proven" to be a quadratic residue.

10. Zero-Knowledge Proofs III: 5/6/14

Today's lecture was given by Mark Zhandry, the TA, since Dan Boneh is out of town.

Recall that the general setup for a zero-knowledge proof is that there is a statement x to prove and a prover P and verivier V who exchange messages; eventually, the verifier either accepts or rejects. For this to be a zero-knowledge proof we require the following.

- The protocol should be *complete*, i.e. if x is true, then V accepts. More formally,
$$\Pr\left[P(x) \leftrightsquigarrow V(x) : V \text{ accepts}\right] = 1,$$
where $P(x) \leftrightsquigarrow V(x)$ symbolizes the interaction between the prover and the verifier.
- The protocol should be *sound*: for all cheating provers P', V rejects false statements x with reasonable probability (usually, but not always, the probability that V accepts in this case is negligible).
- The protocol should be, of course, *zero-knowledge*: in the case where x is true, the verifier learns nothing but the truth of x. More formally, for all cheating verifiers V', there exists a simulator S such that
$$S(x) \approx \text{View}_{V'}[P(x) \leftrightsquigarrow V'(x)].$$
This notation refers to what V' learns.

As an example, consider the Chaum-Pedersen proofs that a tuple is actually a DDH tuple (i.e. $(g, g^a, g^b.g^c) = (g, X, Y, Z)$ such that $c = ab$). Here's a protocol for proving it:

1. The prover chooses a random $s \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$, and sends to the verifier $V = g^s$ and $W = X^s$.
2. Then, the verifier chooses a $t \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$ and sends it to P.
3. Next, the prover sends back $U = s + bt$.
4. The verifier accepts iff $g^U = VY^t$ and $X^U = WZ^t$.

Why is this a zero-knowledge protocol?

- For completeness, if this is really a DDH tuple, then $VY^t = g^s g^{bt} = g^{s+bt} = g^U$, and $WZ^t = g^{as} g^{abt} = g^{a(s+bt)} = g^{aU} = X^U$.
- For soundness, suppose the prover cheats: $V = g^s$ and $W = X^{s'}$ for some s' not necessarily equal to s. After receiving a t, the false prover has to come up with a U that makes the two equations hold, but there is no choice of s, s, and U that make the equations hold. This is because $VY^t = g^{s+bt} = g^U$, so $U = s + bt$ and $WZ^t = g^{as'} g^{ct} = g^{aU}$, so $aU = as' + ct$. Rearranging these a bit, $a(s - s') = (ab - c)t$. But the right-hand side is something nonzero, multiplied by a random element in $\mathbb{Z}_q$, so it's a random element in $\mathbb{Z}_q$, so the probability of the false prover succeeding is negligible (probability $1/q$).
- Finally, we should prove that this protocol is zero-knowledge for verifiers that generate t before seeing V and W. Given some verificiation algorithm V', define a simulator S that chooses $v \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$, and run V' to get t. Then, set $V = g^v/Y^t$ and $W = X^v/Z^t$, and one can show that the transcript of S provides no extra information.[22]

Now, let's look at an application of these Chaum-Pedersen proofs, to prove that two ElGamal ciphertexts encrypt some common message. Recall that this involves choosing a secret key $\text{sk} = z \in \mathbb{Z}_q$ and $\text{pk} = h = g^x$. Then, to encrypt, $\text{Enc}(h, m)$ chooses an $r \overset{\text{R}}{\leftarrow} \mathbb{Z}_a$, and then the ciphertext is $(g^r, h^r \cdot m)$, and decrypting $\text{Dec}(x, c_1, c_2)$ sets $c_2/c_1^x = h^r \cdot m/g^{rx} = m$.

[22]The verifier is described as both cheating and honest: this means that it can choose t arbitrarily so long as it does so before seeing V and W.

How can we prove that two messages have the same plaintext without knowing anything else about them? Suppose $A = g^r$ and $B = h^r \cdot m$, and $C = g^s$ and $D = h^s \cdot m$. Then, $(g, h, A/C, B/D)$ is a DDH-tuple iff they have the same plaintext, because in that case it is $(g, g^x, g^{r-s}, g^{x(r-s)})$. Then, using Chaum-Pedersen, the answer to this question can be discovered.

Zero-Knowledge Proofs of Knowledge. A related kind of proof is the oddly named zero-knowledge proof of knowledge, in which a prover proves that it knows something, without verifying the value of what it knows.

Example 10.1. In lieu of a formal definition, suppose the prover wants to prove that it knows the discrete log of one group element with respect to another (i.e. $h = g^x$, with g, h given), and wants to prove this to the verifier without leaking information.[23] The Chaum-Pedersen proofs are zero-knowledge proofs of knowledge, but there's a simpler one due to Schnorr.

(1) The prover generates an $r \xleftarrow{\text{R}} \mathbb{Z}_q$ and sends $t = gr$ to V (committing to r, in some sense).
(2) Then, the verifier chooses a $c \leftarrow \mathbb{Z}_q$, which can be thought of as a challenge.
(3) The prover sends back $s = r + cx$ to the verifier.
(4) Then, the verifier accepts iff $g^s = th^c$.

For these kinds of proofs of knowledge, soundness isn't as important, but similar completeness and zero-knowledge properties hold.

Here's why this protocol is complete: $th^c = g^r g^{cx} = g^{r+cx} = g^s$, so the verifier correctly detects if the prover did the right thing. As for zero-knowledge, if one picks a random simulation S and runs a malicious verifier V' to get c, the verifier can't see anything that it didn't already have.

We can also show that if V accepts for a cheating prover P', then there exists an extractor E such that $E^{p'} \to x$ (i.e. it's an algorithm that, given access to p', can output the discrete log). More "formally,"' if $\Pr[P' \leftrightarrow V' \text{ accepts}]$ is non-negligible, then $\Pr[E^{p'} \to X]$ is also non-negligible.

Assuming that P' works with probability 1, here's what the extractor actually does. This probabilistic assumption is unnecessary, but makes the logic easier.

(1) First, the extractor runs the cheating prover P' to get the value of t.
(2) Then, it creates a random challenge $c_1 \xleftarrow{\text{R}} \mathbb{Z}_q$ and feeds it to P'.
(3) Then, P' outputs an s_1 such that $g^{s_1} = th^{c_1}$.
(4) Now, rewind P' to after step (1) and give it another challenge $c_2 \xleftarrow{\text{R}} \mathbb{Z}_q$.
(5) Then, P' outputs an s_2 such that $g^{s_2} = th^{c_2}$.

Then, $g^{s_1 - s_2} = h^{c_1 - c_2}$, so $g^{(s_1-s_2)/(c_1-c_2)} = h$, so g is the discrete log.

Zero-knowledge proofs of this form (commitment, challenge, response) are called Σ protocols, maybe because the flow of knowledge back and forth over time looks sort of like the uppercase Σ, but it's all Greek to me.

It is possible to make some of these non-interactive, e.g. by setting the random values to instead be the hash of everything seen so far (by the side computing the value), so the challenge is the hash H of the statement concatenated with the commitment. If H is a random oracle, the same security results hold. For example, in the case of Scnhorr's protocol, the prover first computes $t = g^r$, $c = H(g, h, t)$, and $s = r + cx$. Then, it outputs (t, s), and the verifier checks that $c' = H(g, h, t)$ satisfies $g^s = th^c$.

Signatures. Σ protocols can be tweaked slightly to produce signatures schemes. Consider a signature protocol with statement S, commitment C, and response R. Then, one can define a signature with $c = H(S, C, m)$ and $\sigma = (C, R)$. For example, in the case of Schnorr's protocol, this should look familiar: the key generator is $x \xleftarrow{\text{R}} \mathbb{Z}_q$, the public key is $\text{pk} = h = g^x$, and the secret key is $\text{sk} = x$. To sign, the signer chooses an $r \xleftarrow{\text{R}} \mathbb{Z}_q$ and sets $t = g^r$, $c = H(g, h, t, m)$ and $s = r - xc$. Then, $\sigma = (t, s)$. Then, the verifier can also compute the hash: $c = H(g, h, t, m)$, so it checks that $g^s = th^c$ and accepts iff this is so (i.e. the proof is valid). Schnorr signatures are as secure as discrete log is hard, though we've run out of time for the proof, but it also involves an extractor.

11. Zero-Knowledge Proofs IV: 5/8/14

"Oh my God, there's a whole sea of markers over there!"

Recapping zero-knowledge proofs of knowledge, if $R(x, w)$ is an efficient relation between a problem instance x and a witness w, we can define the associated language $L = \{x \mid \text{there exists a } w \text{ such that } R(x, w) = 1\}$. Then, there's a protocol between a prover $P(x, w)$ and a verifier $V(x)$, required to have the following properties.

(1) The protocol should be complete and sound, i.e. V accepts with probability at least $1/2$ iff $x \in L$.

[23]The verifier must be computationally bounded, so that it doesn't just figure out the solution to the discrete log itself.

(2) It's zero-knowledge: for all verification algorithms V^* there exists a simulator S such that for all $x \in L$, the transcript between the prover and the verifier is computationally indistinguishable to $S^*(x)$.

There's also a much weaker property called honest-verifier zero-knowledge, where the transcripts are only considered for honest verifiers. Formally, there exists a simulator S such that for all honest verifiers V,

$$\{\operatorname{trans}(P(x) \longleftrightarrow V(x))\} \overset{c}{\approx} \{S(x)\}.$$

We also talked about the proof-of-knowledge property: that there exists an extractor E for all provers P^* (including malicious ones) and x such that $[P^*(x) \longleftrightarrow V(x)]$ accepts.

As an example, if G is a group such that $|G| = p$ and $g, h \in G$ are generators of G, then we had zero-knowledge proofs of the Diffie-Hellman language $L = \{(g, , g^\alpha, h^\alpha) : \alpha \in \mathbb{Z}_p\}$, along with the Schnorr protocol for zero-knowledge proof of knowledge $R((g, h), w) = 1$ iff $g^w = h$: the prover chooses an $r \overset{R}{\leftarrow} \mathbb{Z}_p$ and sends $u = g^r$ to V; then, the verifier computes a $c \overset{R}{\leftarrow} [0, \dots, 2^{128}]$ and sends it back. The prover then sends $z \leftarrow r + c \cdot w$, and the verifier checks that $g^z = uh^c$. This is a special case of a more general concept called a Σ protocol, where $(u, r) \leftarrow P_0(\mathrm{sk})$ (generated from a secret key using some protocol) is computed by P, which sends u to V; then, V computes $c \overset{R}{\leftarrow} [1, \dots, \beta]$ and sends it to P. Finally, the prover computes a $z \leftarrow P_1(\mathrm{sk}, r, c)$, and sends it to V, which computes $T(\mathrm{pk}, u, c, z)$ to determine whether to accept or reject.

The Fiat-Shamir protocol allows this to be made non-interactive (i.e. P sends only one thing to V, and V sends nothing to P). Assuming we have a hash function H, one can compute $c \leftarrow H(g, h, u)$ and then send z and u as normal, and that if one appends m, instead calculating $c \leftarrow H(g, h, u, m)$, then the result is a signature scheme ($\operatorname{sig}(m) = [u, z]$), called Schnorr signatures. However, Schnorr patented his algorithm, so a different but related algorithm, called DSA, is the federal standard. This, unlike DSA, has security...

Interestingly, this is noticeably shorter than the DSA signature. We have $[u, z]$, where $u \in G$ and $z \in \mathbb{Z}_p$, so if we work mod p, then u has to be about 2000 bits and z has to be about 256 bits, which is quite long. But Schnorr also applied an optimization that allows one to shrink this signature: since the verifier just computes c from u and doesn't use the rest of the information, it's possible to instead send $[c, z]$, which allows the verifier to recompute $u = g^z/h^c$ and then recompute $\widehat{c} = H(g, h, u, m)$ and check if $c = \widehat{c}$. But since $z \in \mathbb{Z}_p$ and c is of size 128, this is going to only be about 400 bits, while DSA signatures are longer. Good thing Schnorr's patent has expired, right? But since it's not the federal standard, if you use it in a product, you can't sell it to the government (which is part of why nobody uses it).

Theorem 11.1. *Schnorr signatures are existentially unforgeable under CMA, assuming H is a random oracle and the discrete log is hard in G.*

Proof sketch. Given some forger, consider an honest simulator S, and run it on an honest verifier. Choose some generators $\mathrm{pk} = (g, h)$ and send them to the forger; then, it sends back a message m, and run S to get (u, c, z), where c is uniform in $[0, \dots, \beta]$, and send (u, z) to the forger. Then, we can "program" H (the standard random oracle trick; this is OK, because c is uniform) such that $H(g, h, u, m) = c$. Then, if the forger can solve this, then it can break the discrete log problem, and an extractor can (more or less) obtain the discrete log w. ⊠

Witness-Indistinguishable Proofs. As usual, suppose there's a relation $R(x, w)$ and an interactive system $P(x, w) \longleftrightarrow V(x)$.

Definition. The system (P, V) is a witness-indistinguishable proof of knowledge if:

- for all (not necessarily honest) verifiers V^* and for all w_0, w_1 such that $R(x, w_0) = R(x, w_1) = 1$, then
$$\{\operatorname{tran}(P(x, r_0) \longleftrightarrow V^*(x))\} \overset{c}{\approx} \{\operatorname{tran}(P(x, w_1) \longleftrightarrow V^*(x))\}.$$
- (P, V) is zero-knowledge.

This is a pretty weak notion of security: this protocol doesn't uniquely identify the witness, but it does leak information about it. Zero-knowledge proofs leak no information about the witness.

Lemma 11.2. *All zero-knowledge proofs are witness-indistinguishable.*

Proof. By the zero-knowledge property,

$$\{\operatorname{tran}(P(x, w_0) \longleftrightarrow V^*(x))\} \overset{c}{\approx} \{S^*(x)\} \overset{c}{\approx} \{\operatorname{tran}(P(x, w_1) \longleftrightarrow V^*(x))\}.$$ ⊠

Zero-knowledge is a much stronger property than witness indistinguishability. However, we don't believe (this is actually still open) that zero-knowledge is maintained under parallel composition, whereas witness indistinguishability is (i.e. running the protocol in parallel 20 times); this is a simple hybrid argument.

Example 11.3. While any zero-knowledge proof is a witness-indistinguishable proof, it's more interesting to give an example that isn't zero-knowledge. Recall that at least four interactions need to happen in any zero-knowledge proof, so consider the following shorter proof, called an OR proof.

We want to satisfy the property that $R((g, h_0, h_1), w) = 1$ iff $g^w = h_0$ or $g^w = h_1$, so consider the following Schnorr-line protocol: if $g^w = h_0$, then choose an $r_0 \overset{R}{\leftarrow} \mathbb{Z}_p$ and $u_0 \leftarrow g^{r_0}$. Then, inside the simulator, set $u_1 \leftarrow g^z/h_1^c$, $c_1 \overset{R}{\leftarrow} \mathbb{Z}_p^*$, and $z_1 \overset{R}{\leftarrow} \mathbb{Z}_p$; then, it sends u_0 and u_1 to the verifier $V(g, h_0, h_1)$, which sends a $c \overset{R}{\leftarrow} \mathbb{Z}_p^*$ back to the prover, which sets $c_0 = c/c_1$. Finally, the prover sends $z_0 \leftarrow r_0 + wc_0$, and sends c_0, z_0, z_1 to the verifier, which checks that for $b = 0, 1$, $g^{z_0} = u_b h_b^{c_b}$, where $c_1 \leftarrow c/c_0$.

If instead $g^w = h_1$, then switch the order; these are called OR proofs because one or the other of the discrete logs is proved, but not both.

This protocol has lots of properties.

- It's a proof of knowledge, in which you should extract some witness, because the extractor checks that if $g^z = uh^c$ and $g^{z'} = uh^{c'}$, then $g^{(z-z')/(c-c')} = h$, so as long as $c \neq c'$, then this works. This is thus generalized to allow the extractor to extract w_0 if the prover has w_0, or w_1 if it has w_1. Thus is independent of the witness, so this works.
- It's also witness-indistinguishable. The transcript is in both cases $(u_0, u_1, c, c_0, z_0, z_1)$, but this isn't computationally distinguishable: u_0 and u_1 are random elements in G, c is generated by a possibly malicious verifier, c_0 is random in $\mathbb{Z}_p^*$ in either case (though if we have to quotient then it's slightly more complicated), and z_0 and z_1 are computed to pass the verification tests, so they're the same in both cases. Thus, this transcript has the same distribution in both cases.
- It's also honest-verifier zero-knowledge, because the transcript can be simulated, where $c \overset{R}{\leftarrow} \mathbb{Z}_p^*$ instead. However, it's not yet zero-knowledge (because it's only got three rounds). By adding a commitment to c, it becomes zero-knowledge, but then it's not a proof of knowledge, because the extractor can't go back and undo the commitment.

Here's an "application" to identification protocols, which is sound but more complicated than other protocols for the same result. However, the idea is still worth knowing for other applications.

An identification protocol is a way for a prover to prove its identity to a verifier (e.g. a car door which unlocks once a specific person has proven its identity). The goal is to protect against eavesdropping security, where an attacker can see many honest interactions between a prover and a verifier, but then cannot fool the verifier.

Let w be the secret key, held by the prover, and g^w be the public key, held by the verifier. It's possible to set up this protocol with challenge-response or even just one-time passwords, But there's a protocol secure against eavesdropping attacks based on an honest-verifier zero-knowledge proof of knowledge: if one has an attacker than can fool the verifier, then the honest interactions it watched can be simulated, and then the extractor can be used to get the attacker to break the discrete log.

Active security is more interesting: the attacker can interact with the prover (e.g. a fake ATM in the mall), and then the attacker can masquerade with the prover as the verifier. One-time passwords aren't actively secure, because one could collect a one-time password and feed it to the verifier. However, challenge-response protocols are secure here (relying on an underlying secure signature scheme).[24]

Proposition 11.4. *Witness indistinguishability implies identification protocols secure against active attackers.*

Proof. Consider a proof system where the prover P has (g, h_0, h_1, w), and the verifier V has (g, h_0, h_1). Then, they execute the witness-indistinguishable OR protocol as before.

Suppose there exists an active attacker that can defeat this protocol; then, we will be able to use it to break the discrete-log problem. We are given as input g and $h = g^\alpha$, and choose h_0 and h_1, so the public key is (g, h_0, h_1) (which the attacker of course has access to). The trick will be to choose a random bit $b \overset{R}{\leftarrow} \{0, 1\}$ and a random $r \overset{R}{\leftarrow} \mathbb{Z}_p$, and let $h_b \leftarrow g^{r b_b}$ and $h_{1-b} \leftarrow h$, so we know the discrete log of exactly one, but the attacker doesn't know which. Then, the attacker makes a number of probes to try and determine the secret key.

Since we know one of the discrete logs, we don't need a simulator, and thus don't need zero-knowledge.

Now, the attacker tries to fool the verifier, but since it's a proof of knowledge, we can extract a witness… and there's a 50% probability that it's a witness for h_b, which isn't particularly helpful (so abort). Otherwise, we get the discrete log of h, which is α, so the attacker can calculate discrete logs with probability $1/2$. ⊠

[24]There's another variant of challenge-response protocols using encryption, where the verifier sends a random encryption of a message m, and the prover decrypts this message. It turns out the security property needed is CCA_1 (or if that's confusing, chosen ciphertext security will do).

Notice that witness-indistinguishability is crucial so that the attacker can't tell which witness is which, so it can't force failure. Thus, there's a three-round active-ID protocol (i.e. secure against active attacks), and the security is based on the discrete log, and no random oracles.

In the real world, of course, we can build signatures with a two-round protocol. However, without random oracles, we don't now how to build them on the discrete log, and instead need stronger assumptions. Does the difference matter in practice? Maybe not.

12. Elliptic Curves: 5/13/14

". . . and there's no political subtext to labeling Democrats 0 *and Republicans* 1*."*

For an application of proving relations with zero-knowledge, consider ElGamal encryption: suppose we have a public key $\mathrm{pk} = (g, h)$ and $|G| = p$. Then, let $c_0 = (g^{\alpha_0}, h^{\alpha_0} \cdot g^{m_0})$, $c_1 = (g^{\alpha_1}, h^{\alpha_1} \cdot g^{m_1})$, and $c_2 = (g^{\alpha_2}, h^{\alpha_2} \cdot g^{m_2})$; then, the goal is to show that $m_2 = m_1 \cdot m_0$.

Proving addition (that is, $m_2 = m_1 + m_0$) is easy, since we already have proof for Diffie-Hellman, so check this on

$$\left(g, h, \frac{c_0[1] \cdot c_1[1]}{c_2[1]}, \frac{c_0[0] \cdot c_1[0]}{c_2[0]}\right),$$

which forms a DDH tuple iff $m_2 = m_1 + m_0$, so that the third term is $h^{\alpha_0+\alpha_1-\alpha_2}$ and the fourth is $g^{\alpha_0+\alpha_1-\alpha_2}$, which uses a Chaum-Pederson proof.

The proof for multiplication is much more involved, but has a nice application to voting protocols. Here, each user chooses a $v_i \in \{0, 1\}$, and sends $E(\mathrm{pk}, v_1) = (g^{\alpha_i}, h^{\alpha_i} \cdot g^{v_i})$. Then, the center can compute

$$c^* = \left(\prod_{i=1}^{n} c_i[0], \prod_{i=1}^{n} c_i[1]\right),$$

which after decrypting is $g^{\sum v_i}$. But since there aren't that many votes, then the discrete log can be taken, yielding $\sum v_i$, which provides the result.

It's a little uncomfortable that the center can decrypt individual users' votes, but this can be alleviated by distributing the key amongst several shared trustees. More problematic is that as is, this protocol has no protection against the user ecrypting 30, or -12, or anything like that to make their vote count more than others'. Thus, along with its vote, a user should submit a zero-knowledge proof that its vote is eihter zero or one.

The most efficient way to do this is an OR proof: if $c = (g^\alpha, h^\alpha \cdot g^m)$, one can prove that one of $(g, h, c[0], c[1])$ or $(g, h, c[0], c[1]/g)$ is a DDH-tuple, so this uses the standard zero-knowledge proof. An alternate proof is to prove that c is an encryption of m^2, and then feed three copies into the multiplication checker. This tests that $m^2 = m$, which is only true for $m = 0$ and $m = 1$.

The History of Elliptic Curves. Moving on to elliptic curves: one could (and many do) spend one's whole life studying elliptic curves. We'll start with a history of why they're interesting in general, and then why they're interesting for crypto specifically.

Elliptic curves combine all three branches of mathematics: algebra, analysis, and geometry, in a way that needs all of them. The history for algebra starts a pretty long time ago, in the days of Diophantus (in Alexandria, 200 AD). He wrote seven volumes called *Arithmetica*: they were stored in the Library of Alexandria for centuries, disappeared, and then four volumes popped up again in the Vatican library a millenium later. Probably by accident, someone had grabbed books from the Library of Alexandria, when it was burning or maybe earlier. One of these four was then lost at a later date. . .

Diophantus started by asking about rational solutions to the equation $x^2 + y^2 = 1$ (first asked by the Pythagoreans), which leads to integer solutions to $x^2 + y^2 = z^2$. He may have been the first mathematician to think of fractions as numbers. Then, he generalizes to other conics, e.g. $ax^2 + by^2 + cxy + dx = 0$, with $a, b, c, d \in \mathbb{Z}$. For $ax^2 + by^2 = 1$, if one has a solution $y = \widehat{a}x + \widehat{b}$ (i.e. one of its two solutions is rational), then the other root must also be rational. This means that if one intersects the conic with any line through that rational point, then the other intersection are also rational (though there are some conics without rational points, e.g. $x^2 + y^2 = 3$). Thus, there's an easy map from the rational numbers to rational points on conics, and it works in every field.

Then, he looked at problems such as $x^3 + y^3 = z^3$, or really, what are the rational points on $x^3 + y^3 = 1$? Fermat said something about this later. Diophantus chose a really arbtrary-looking cubic $y(6 - y) = x^3 - x$. More generally, the goal is to understand rational points on curves of type $y^2 = x^3 + ax + b$, for $a, b \in \mathbb{Q}$ (i.e. the zero set of $f(x, y) = y^2 - x^3 - ax - b$).

There are two special cases of note.

- If one views the curve as the zero set of a curve $f \in \mathbb{Q}[x,y]$, then a double point is a point (x_0, y_0) such that

$$f(x_0,y_0) = \frac{\partial f}{\partial x}(x_0,y_0) = \frac{\partial f}{\partial y}(x_0,y_0) = 0.$$

  This can be realized as a place where a curve intersects itself, as in Figure 1. But then, by drawing a line with rational slope through a double point, one obtains another rational point, providing a bijection between $\mathbb{Q}$ and the rational points on the curve. In particular, $y^2 = x^3 + ax + b$ has a double point iff $x^3 + ax + b = 0$ has a double root (which can be found by checking whether discriminant $4a^3 + 27b^2 = 0$). If this is the case, then the cube is as difficult as a line (known as the genus zero case; as we saw, all conics are degree zero). The genus explains its complexity, and isn't correlated with the degree.

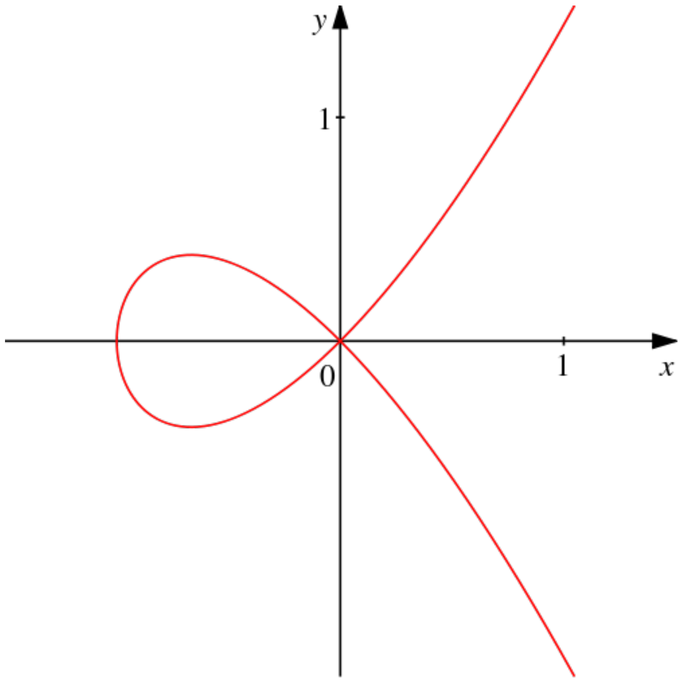

FIGURE 1. A cubic curve with a double point. Source: `http://en.wikipedia.org/wiki/Cubic_plane_curve`

- If the curve has no double roots, as when $4a^3 + 27b^2 \neq 0$, then the line through any two rational points usually intersects a third point, which is also a rational point. Then, it can be flipped across the y-axis, and this point with one of the others gives another point, and so on... This operation suggests a sort of group law, but there's no reason just yet to think that it satisfies the group axioms.

Introduce the notation $E/\mathbb{F}$ for a field $\mathbb{F}$ to mean such a curve given by $f(x,y) = 0$, where $f \in \mathbb{F}[x,y]$, and $E(\widehat{\mathbb{F}})$ to mean the points of E including the point at infinity.

Poincaré was able to show that $E(\mathbb{Q})$ is a group, and ten years later Hasse showed that it's finitely generated, and of the form $E(\mathbb{Q}) = T \times \mathbb{Z}^n$, where T is a finite torsion group. This is only interesting when the discriminant is nontrivial.

At the same time, there was a parallel development in analysis, completely independent. In the 18th Century, people were very interested in calculating the arc length, e.g. of ellipses. This reduces to an integral: specifically, on the arc 0 to x is

$$f(x) = \int_0^x \frac{\mathrm{d}t}{\sqrt{t^3 + at + b}}.$$

This is known as an elliptic integral (since it was used to calculate length on ellipses, which is where the term "elliptic curves" came from). These integrals aren't solvable in terms of elementary functions, but Euler noticed that given $f(x_0)$ and $f(x_1)$, it's possible to compute $f(x_0 + x_1)$ — though the geometric meaning was lost on him and eventually discovered by Legendre.

Recall that

$$\arcsin x = \int_0^x \frac{\mathrm{d}t}{\sqrt{t^2 - 1}},$$

so this similar-looking integral is the inverse of a periodic function. Jacobi and Abel viewed this as a complex function, and looked at its inverse $f^{-1} : \mathbb{C} \to \mathbb{C}$. They managed to show that it is a doubly periodic function; that is, there are $\omega_1, \omega_2 \in \mathbb{C}$ such that for all $x \in \mathbb{C}$, $f^{-1}(\omega_1 + x) = f^{-1}(\omega_2 + x) = x$. Thus, the function is the same on these lattices, and thus, it's possible to define f^{-1} on a fundamental domain and extend it to all of $\mathbb{C}$. The fundamental domain is a torus, which is genus 1 topologically (which is where the genus term above came from).

It turns out that Weierstrass defined a function called the $\wp$ ("p" or "pe") function $\wp : \mathbb{C} \to \mathbb{C}$, such that all doubly periodic functions with given periods ω_1 and ω_2 are polynomials in $\wp$. (In LaTeX, this is written `\wp`.) This function is complicated to write down, but satisfies the differential equation

$$\wp'(z)^2 = 4\wp(z)^3 - g_2\wp(z) - g_3$$

for $g_2, g_3 \in \mathbb{C}$. But wait, this is exactly the degree-3 equation we were looking at before!

The doubly periodic relation induces a lattice L, so the fundamental domain is $\mathbb{C}/L$. Thus, consider the curve $\{(x, y) \in \mathbb{C} \mid y^2 = 4x^3 - g_2x - g_3\}$; then, the map $z \in \mathbb{C} \mapsto (\wp(z), \wp'(z))$ sends $\mathbb{C}/L \to E(\mathbb{C})$. This map preserves a kind of addition: the addition rule in $\mathbb{C}$ is sent to the chord-and-tangent method of adding on elliptic curves, where two points are added by taking the line through them, then the third point it intersects, and then reflects it across the y-axis.

Not every line intersects the curve at three points, so lines that only intersect at two points are said to also intersect a point at infinity (which can be made rigorous): for example, for a point p, $-p$ is its reflection across the y-axis, and then $p + (-p)$ intersects the curve twice. Thus, the sum is that point at infinity, which is the identity (and thus is the image of 0 under the map above). (Since we're mapping out of the fundamental domain, which is a torus, there are three points of order 2 on the curve, which are exactly where it intersects the x-axis, since the curve has vertical tangent here. These points come from the points on the fundamental domain which have order 2. Really, thinking about elliptic curves as maps from the fundamental domain is very useful.) Thus, $E(\mathbb{C}) = \{(x_0, y_0) \mid y_0^2 = x_0^3 + ax_0 + b\} \cup \{0\}$ (since the point at infinity is the identity, it's often written 0 instead of ∞).

The $\wp$ function is easy to evaluate, but hard to invert, which should be suggestive...

The curve addition law can be explicitly written down: if $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are distinct, then let their sum be $P_3 = P_1 + P_2 = (x_3, y_3)$ be given by $x_3 = s^2 - x_1 - x_2$ and $y_3 = y_1 + s(x_3 - x_1)$, where $s = (y_2 - y_1)/(x_2 - x_1)$. In particular, this addition sends rational points to rational points.[25] A similar formula exists when $P_1 = P_2$.

But since this sends rational points to rational points, we are perfectly able to do this over a finite field: choose $a, b \in \mathbb{F}_p$, so that $E_{a,b}(\mathbb{F}_p) = \{(x_0, y_0) : y_0^2 = x_0^3 + ax_0 + b\} \cup \{0\}$, with the same addition laws, giving a very nice, finite abelian group. Thus, cryptographers begin asking how hard the discrete log is on this group, and it turns out that it's pretty hard. For a size bound, in 1942, Weil proved from jail that for all $a, b \in \mathbb{F}_p$, $|E(\mathbb{F}_p)| = p + 1 - t$, for an error term $|t| < 2\sqrt{p}$.

13. Elliptic Curves and Pairings: 5/15/14

"Is Miller still around?" "Yes, he works at a three-letter agency... the moral of the story is, if at first you don't succeed, apply for a job at the NSA."

Recall that an elliptic curve over a field $\mathbb{F}$, denoted $E/\mathbb{F}$, is the curve $y^2 = x^3 + ax + b$ for $a, b \in \mathbb{F}$ and such that the discriminant $4a^3 + 27b^2 \neq$. Then, the point set is $E(\widehat{\mathbb{F}}) = \{(x, y) \in \widehat{\mathbb{F}}^2 \mid y^2 = x^3 + ax + b\} \cup \{0\}$. Over $\mathbb{C}$, an elliptic curve is determined by doubly periodic functions with periods 1 and ω, so the curve is given on a fundamental domain. Then, the transformation $z \mapsto (\wp(z), \wp'(z))$ sends every point in the fundamental domain to a point on E, and sends addition to the chord-and-tangent method for additon on the elliptic curve. The lattice and fundamental domain are two-dimensional, and so the elliptic curve group is two-dimensional: it's the product of two cyclic groups.

Over a finite field, the same story happens: we take $a, b \in \mathbb{F}_q$ such that $4a^3 + 27b^2 \neq 0$ in $\mathbb{F}_q$.

If q is prime, $\mathbb{F}_q$ is just the field of integers modulo q (with addition and multiplication mod q). But then one can write $\mathbb{F}_q \subseteq \mathbb{F}_{q^\alpha}$, which is the field of polynomials over $\mathbb{F}_q$ and modulo polynomial of degree α. This is a degree-α extension of $\mathbb{F}_q$: $|\mathbb{F}_q| = q$, but $|\mathbb{F}_{q^\alpha}| = q^\alpha$. However, $\left|\mathbb{F}_q^\times\right| = q - 1$ and $\left|\mathbb{F}_{q^\alpha}^\times\right| = q^\alpha - 1$.

So now we can talk about elliptic curves over $\mathbb{F}_{q^\alpha}$: $E_{a,b}(\mathbb{F}_{q^\alpha}) = \{(x, y) \in \mathbb{F}_{q^\alpha} : y^2 = x^3 + ax + b\} \cup \{0\}$. This has a group law $\boxplus$, which is the "chord-and-tangent method," but not really, since t has no geometry (just the same equations): if $p_1, p_2 \in E_{a,b}(\mathbb{F}_{q^\alpha})$, then $p_1 \boxplus p_2 \in E_{a,b}(\mathbb{F}_{q^\alpha})$.

Theorem 13.1 (Weil 1942)**.** *If $p = |E_{a,b}(\mathbb{F}_q)|$ (where $4a^3 + 27b^2 \neq 0$), then $p = q + 1 - t$, where $|t| < 2\sqrt{q}$.*

This value t is called the trace. The point is that $p \approx q \pm 2\sqrt{q}$. For example, on $y^2 = x^3 + 1$, then $q \equiv 2 \pmod 3$, so $x \mapsto x^3$ is one-to-one (in crypto terms, 3 is a valid RSA modulus). Thus, for any y, square and subtract 1, and

[25] It's possible to prove that this is a group directly from these group laws, but it takes about ten pages of computation! Nine and a half pages too many, since the group structure is induced from $\mathbb{C}$.

thus there's one value of x: thus, there's one x for each y, plus the point at infinity, so $p = q + 1$. Thus, the trace is zero (sometimes, this is called a trace-zero curve).

Claim. If $p = |E(\mathbb{F}_q)|$, then the sequence $|E(\mathbb{F}_q)|, |E(\mathbb{F}_{q^2})|, |E(\mathbb{F}_{q^3})|, \ldots$ is completely determined by p (once we have t and q), and is in fact given by a nice generating function. This is useful for generating elliptic curves with a given number of points: start by brute-forcing the number of points on a curve over $\mathbb{F}_q$, and then extend it to some $\mathbb{F}_{q^\alpha}$.

Claim. Suppose that $p \nmid q(q-1)$ (so we're ignoring a few degenerate cases) and p is prime (so that $E(\mathbb{F}_q)$ is cyclic), and let $\alpha > 0$ be the smallest integer such that $p \mid q^\alpha - 1$. Then, when we go to the extension, $\mathbb{Z}_p^+$ (additive group of integers mod p) appears twice: $\mathbb{Z}_p^+ \times \mathbb{Z}_p^+ \subseteq E(\mathbb{F}_{q^\alpha})$. One can sort of see this as a $p \times p$ lattice of order-p points in this extension. This is no coincidence: it's the same two-dimensionality as that from the complex numbers.

Now, thinking as cryptographers, here's a new large group. So, how hard is discrete log? For random $a, b \in \mathbb{F}_q$, the best-known discrete-log algorithm runs in time approximately $O(\sqrt{q}) = o(e^{(1/2)\log q})$. For 128-bit security, one needs $q > 2^{256}$, and for 256-bit security, one wants a 512-bit modulus. Compare this to working in $\mathbb{F}_q^*$, where the best algorithm has approximate running time of $O(e^{\sqrt[3]{\log n}})$, which grows a lot because of the cube root! For 128-bit security, one needs $q > 2^{2048}$, and for 256-bit security, one needs upwards of 2^{15000}, which is icky. Thus, the world is moving towards elliptic curve cryptography, and specifically a curve called P256 (over 99% of large websites using elliptic curves). How its parameters were obtained is a matter of great speculation, but we believe it to be secure: someone obtained random numbers and fed them through a SHA-256 hash to get the parameters.[26] The discrepancy in eliptic-curve discrete-log and $\mathbb{F}_q^*$ discrete-log is a twenty-year open problem.

For example, here's the elliptic Diffie-Hellman protcol. Alice and Bob each respectively choose an $a \overset{\mathrm{R}}{\leftarrow} \{1, \ldots, |\mathrm{P256}|\}$ and a $b \overset{\mathrm{R}}{\leftarrow} \{1, \ldots, |\mathrm{P256}|\}$. Then, Alice sends $A \leftarrow a \cdot P$ to Bob (this $\cdot$ means repeated doubling), and Bob sends $B \leftarrow b \cdot P$. Then, each of Alice and Bob has $ab \cdot P$, so they can apply the key derivation function to it.

However, not all curves are strong: if $|E(\mathbb{F}_q)| = q$, then the trace is $t = -1$. This means that $\wp$ is invertible, so in some sense, one can push it back to $\mathbb{C}$, where it goes by the name of division.

The equation we saw for elliptic curves is known as the Weierstrass form, but there's another form called the Edwards form (discovered as recently as 2007!), given by $x^2 + y^2 = c^2(1 - x^2y^2)$. See Figure 2 for a picture over $\mathbb{R}$. Then, the addition algorithm is very fast and easy; there are no cases:

$$(x_1, y_1) \boxplus (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{c(1 + x_1x_2y_1y_2)}, \frac{y_1y_2 - x_1x_2}{c(1 - x_1x_2y_1y_2)} \right).$$

Thenm the point at infinity is just $(0, c)$, since the curve itself is bounded.

This is currently used in the 25519 curve software, which is used by Google on Android devices. It's not a major security cost or anything, just arithmetic simplification.

Pairings. The real power of elliptic curves is that they have additional structure, such as the notion of a pairing.

Definition. Let G_0, G_1, G_T be finite groups of prime order p. Then, a pairing is a map $e : G_0 \times G_1 \to G_T$ such that:

(1) e is bilinear, i.e. for all $g_0 \in G_0$ and $g_1 \in G_1$ and $a, b \in \mathbb{Z}$, $e(g_0^a, g_1^b) = e(g_0, g_1)^{ab}$.
(2) e is nondegenerate, i.e. there exist $g_0 \in G_0$ and $g_1 \in G_1$ such that $e(g_0, g_1) \neq 1$.
(3) e must be efficiently computable.

Here, G_T stands for "target group;" it's the output of the pairing.

The simplest application is something known as BLS signatures. Suppose one has a hash function$H : \{0,1\}^* \to G_0$. Then, the public key is $\mathrm{pk} = (g_1, g_1^\alpha)$, and the secret key is α. Then, the signature is $\mathrm{sig}(m) = H(m)^\alpha \in G_0$ (so this is a short signature!), and the verification algorithm, given g_1, g_1^α, m, and the signature σ tests whether $e(g_1, \sigma) = e(g_1^\alpha, H(m))$, since σ ought to be $H(m)^\alpha$ and the pairing is bilinear, so the exponent can be moved around.

Example 13.2. Let $p = |E(\mathbb{F}_q)|$, where $p \mid q^\alpha - 1$ and $\alpha = 6$. Then, $\mathbb{Z}_p \times \mathbb{Z}_p \subseteq E(\mathbb{F}_{q^\alpha})$, so there's an $\omega \in \mathbb{F}_{q^\alpha}$ such that $\omega^p = 1$. Thus, let $G_T = \langle \omega \rangle$; then the Weil pairing is the map $e_p : G \times G \to G_T$. An algorithm due to Miller allows this pairing to be computed in time $o(\log^3 p)$.[27]

[26] Suppose there were a faster algorithm that worked on one elliptic curve in a million. Then, it is reasonable to try to reverse the hash to get something which leads to a special curve. Maybe this happened. Who knows?

[27] Weyl came up with his pairing in World War 2, but in the early '80s, someone published a paper claiming that the pairing is a polynomial with exponential degree, and thus not efficiently computable. So Miller found this algorithm and wrote a paper, but got a lot of rejections from people who claimed it had no applications... and now, of course, it has about 10,000 citations!

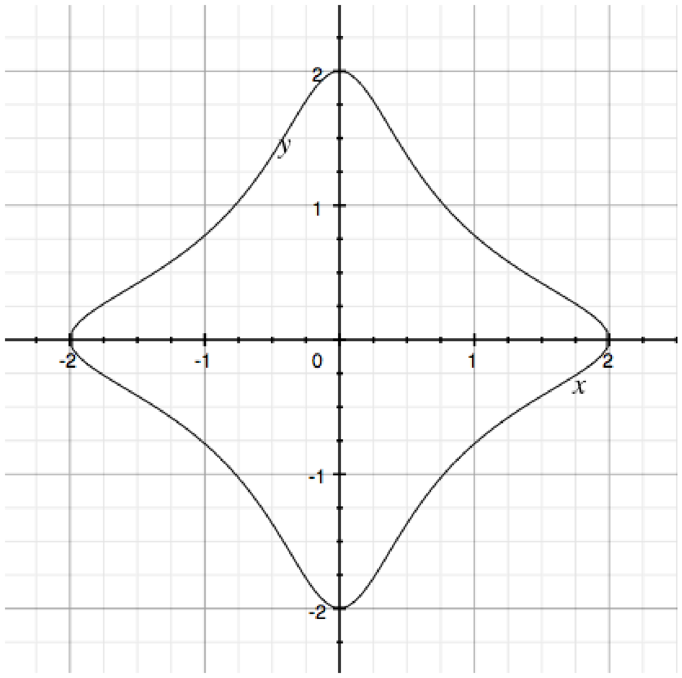

FIGURE 2. The Edwards form of the elliptic curve $x^2 + y^2 = 4(1 - x^2y^2)$.

Note that pairings in general are very rate; there's a whole subfield of crypto now for constructing curves with pairings.

Thinking of the elliptic curve as a $p \times p$ grid, because of the inclusion $\mathbb{Z}_p \times \mathbb{Z}_p \subseteq E(\mathbb{F}_q)$, is generally a good idea. In particular, one can obtain a basis $\{P_1, P_2\}$ of G, so by basic linear algebra, given Q_1 and Q_2 in G, there exist a, b, c, d such that

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \end{pmatrix} = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}.$$

Then, $e(Q_1, Q_2) = \omega^{ad-bc}$, i.e. to the power of the determinant! The Weil-Miller magic here is to compute the pairing without knowing $a, b, c,$ or d. This is an alternate[28] viewpoint for the pairings and algorithms.

Generalizing this to k-linear maps for $k > 2$ is a very open question. These maps are given by extending the pairing definition, such that the discrete log is still hard. It first seems like a problem that we don't have $\mathbb{Z}_p^3 \subseteq E(\mathbb{F}_{q^\alpha})$, but this can be surmounted by using genus-2 curves. However, the extension of Miller's algorithm doesn't work, and there might not be a way to compute this in the general case without the matrix entries; in some sense, there's something magical about 2.

To construct pairings more generally, let $\mathcal{F} = \{f : X \to Y\}$ be a family of functions (sometimes called operators). Then, a pairing $e : X \times \mathcal{F} \to Y$ can be very naturally constructed by function evaluation: $e(x, f) = f(x)$. This means that pairings are very natural in many places in mathematics, but trilinear maps aren't.

To make this work in elliptic curves, it's necessary to discuss the notion of functions out of elliptic curves. This involves algebraic geometry, which is difficult to learn out of abstract books but isn't so bad in these cases.

Definition. A divisor is a formal sum of a finite number of points on on $E(\mathbb{F})$:

$$\mathcal{A} = \sum_P a_P \cdot (P).$$

For example, we could have $\mathcal{A} = 2 \cdot (P_1) + 3 \cdot (P_2) - 5 \cdot (P_3)$. We will only consider divisors such that $\sum_P a_P = 0$.

A function on the elliptic curve $E(\mathbb{F})$ is a rational function in the two variables: $f \in \mathbb{F}(X, Y)$ (the ratio of two polynomials), e.g. $f(x, y) = (x^2 + y)/(x + 3)$, but where two functions f_1 and f_2 are called equivalent if there exists an $h \in \mathbb{F}(X, Y)$ such that $f_1(x, y) = f_2(x, y) + h(x, y) \cdot (y^2 - x^3 - ax - b)$. In other words, we identify functions that are equal on the curve. This is called the field of functions on this curve, and has a wealth of structure.

[28]Or maybe an alternating viewpoint. Heh.