Scribe Notes

Mathematical Foundations of Cryptography

Math 267A - UCSD, Winter 1997

Instructor: Sam Buss

Contents

Math 267a - Foundations of Cryptography

Lecture #1: 6 January 1997

Lecturer: Sam Buss

Scribe Notes by: Jeremy Martin

1 Administrivia

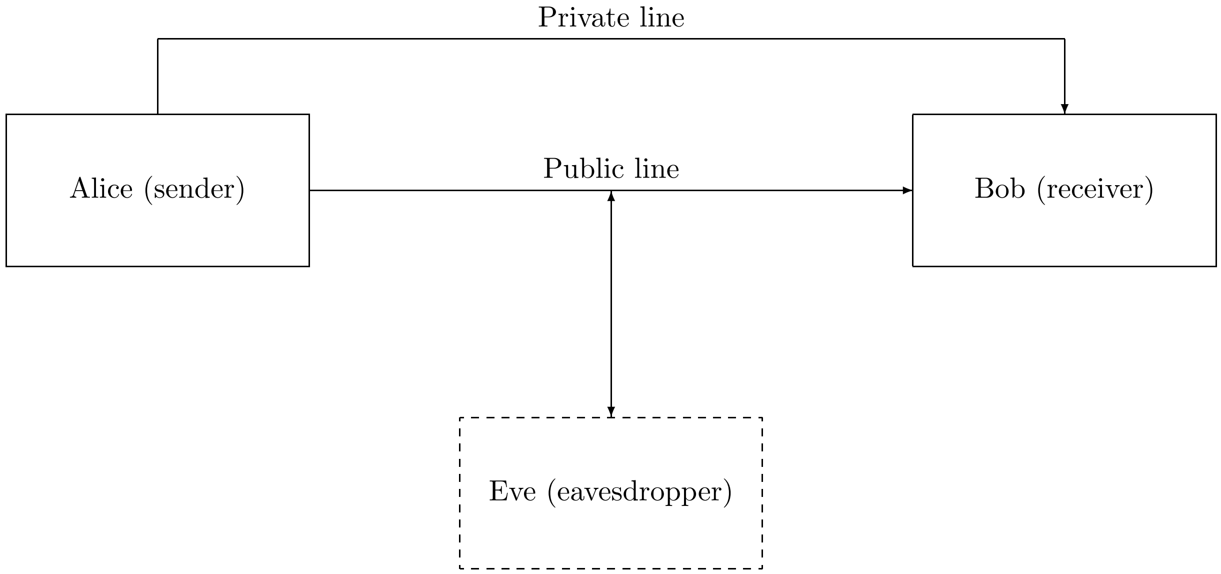
Texts for the course: (in order of relevance to this course).

1. M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton U. Press, 1996.
2. O. Goldreich, *Foundations of Cryptography*, unpublished manuscript (available electronically at **http://www.eccc.uni-trier.de/eccc-local/ECCC-Books/oded_book_readme.html**)
3. A, Menezes, P. van Oorshot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
4. N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 1994.
5. B. Schneider, *Applied Cryptography*, Wiley, 1996.

The Math 267a home page is at: **http://euclid.ucsd.edu/~sbuss/CryptoCourse/**.

2 The Cryptologic Model

Some terminology: **Cryptography** refers to making up codes, **Cryptanalysis** to breaking codes, **Cryptology** to both.



Alice and Bob's goal is to send a message m, consisting of n bits, along the public line while preventing Eve from obtaining any information other than the fact that the message was sent.

"One-time pad": At some time in the past, Alice and Bob used the private line to agree on a random n-bit key k. To send the message m, Alice encodes it as

$$e = m \oplus k$$

(where $\oplus$ denotes bitwise addition modulo 2, or the "parity" operation). Then e is the message that Alice sends to Bob via the public line. Bob, knowing the key, can decrypt it as $m = e \oplus k$, but Eve cannot recover m without knowing k.

The problem with this method is that the keys are not reusable — if Alice and Bob can keep picking new random keys via the private line, they may as well use the private line to communicate anyway!

Suppose that Alice uses the same key k to encode two messages m_1, m_2 as $e_1 = m_1 \oplus k$, $e_2 \oplus k$. Then Bob can decode both messages correctly, but Eve can compute

$$e_1 \oplus e_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

If m_1 and m_2 are normal English text, then this should be enough for Eve to break the code. Even if not, it is information that Alice and Bob don't want Eve to have.

Note that we are assuming that Eve knows the basic *algorithms* that Alice and Bob are using, but not the information that they exchange via the private line — i.e., the key. The reason for this is that we are not usually talking about a single Alice and Bob, but want a single method of encryption which will work for anyone who wants to share data securely.

3 Random and Pseudorandom Keys

Let g be a function where $g(k,i) = k_i \in \{0,1]^n$ (i.e., a word of n bits). Alice can use the k_i as codewords to encrypt messages $m_1, m_2, \ldots$ as

$$e_1 = m_1 \oplus g(k,1)$$
$$e_2 = m_2 \oplus g(k,2)$$
$$\vdots$$

Alice desires g to be a *pseudorandom function*. Informally, this means that although g selects keywords according to some predetermined algorithm or distribution, there exists no computational way to distinguish them from "truly random" keywords. If g is pseudorandom, then even if Eve knows e_1 and e_2, there exist 2^n possible message pairs (m_1', m_2') that could have been sent (since $m_1 \oplus m_2 = e_1 \oplus e_2$).

So Alice is hoping that Eve doesn't have enough computational power to figure out any information about the encoded messages by brute force. For example, if $n = 128$, then 2^n equals 10^{13} times the age of the universe in nanoseconds — making the problem not only intractable with modern computing power, but also intractable with likely future computing capabilities.

4 When Are Problems "Hard"?

Usually, our cryptologic theorems will take the form

"If Problem A is hard to solve, then Problem B is also hard to solve."

Our proofs will generally be constructive and will show that if we are given a method of solving Problem B efficiently, then we can obtain a method for solving Problem A efficiently.

For our best theorems, Problem A will be the problem of factoring large integers. In general, we will be assuming at least that $P \neq NP$.

Definition: An algorithm is *polynomial-time* iff there exists a polynomial $p(n)$ such that for all inputs of n bits, the algorithm halts in at most $p(n)$ steps.

(Notice that the key-search problem requires 2^n steps, which is not polynomially bounded.)

Correspondingly, a function is defined to be *polynomial-time* if there exists a polynomial-time algorithm to compute it. Usually we will be talking about functions of the form

$$f : \{0,1\}^* \to \{0,1\}^*$$

where $\{0,1\}^*$ denotes the set of all finite strings of 0's and 1's, i.e.,

$$\bigcup_{n \in \mathbf{N}} \{0,1\}^n$$

Definition: A *predicate* is a subset of $\{0,1\}^*$. A predicate is said to be *polynomial-time* iff its characteristic function is polynomial-time.

Definition: P is the set of polynomial-time predicates.

Definition: NP is the set of "nondeterministic" polynomial-time predicates. Formally, $L \in NP$ iff there exists some $L' \in P$ and polynomial $l(n)$ such that for all $x \in \{0,1\}^*$,

$$x \in L \iff \left[\exists y \in \{0,1\}^* : \ (x,y) \in L'\right]$$

where (x,y) is some one-to-one encoding of x and y as a single string (for example, by alternating bits).

Notice that we need to define the computational models we are using — for example, Turing machines or RAMs (random-access machines). For the purposes of this course, we will not specify a precise model of computation; rather, but one should think of computations as being carried out on an idealized computer. The computer is idealized in that it does not have any fixed bounds on the amount of time and memory space which may be used. The computation time, or the number of steps in a computation, should be measured in terms of the number of bit-operations performed by the idealized computer.

Since we are not able to prove any unconditional negative results about computability, we will not be hampered by the fact that we have not picked a particular computational model. Instead, most of our proofs will involve the explicit construction of algorithms.

Math 267a - Foundations of Cryptogaphy

Lecture #2: 13 January 1997

Lecturer: Sam Buss

Scribe Notes by: Robert Ellis

5 P, NP, and Feasibility

From the first lecture, we have seen that a predicate L is a subset of Σ^*, and that two types of predicate classes are P and NP, whose characteristic functions are polynomial-time and nondeterministic polynomial-time, respectively. Perhaps the most notorious unsolved problem in algorithmic complexity theory today is whether $P = NP$ (the forward containment is straightforward).

Example. Let $COMPOSITES$ be the predicate of all composite positive integers in binary notation. As nondeterminism allows "guessing" in the characteristic function of a predicate, we can construct a nondeterministic polynomial-time algorithm which guesses a nontrivial factor of the composite in question, and so $COMPOSITES$ is in NP.

Also of interest is whether or not decidability of membership in a given predicate is computationally "feasible". Loosely, feasibility is a measure of confidence in obtaining results in a reasonable amount of time, for example, in determining whether a given member of Σ^* is in a predicate. That is to say algorithms that use an excessively large amount of time will be deemed "infeasible", whereas algorithms which can be run in reasonable "do-able" amount of time will be viewed as "feasible". Polynomial-time algorithms, such as characteristic functions of predicates in the class P, are deemed feasible, but exponential-time algorithms are deemed infeasible.

One idea for extending the class of feasible predicates is to incorporate randomization into algorithms. In general a feasible randomized algorithm should make "coin-flips" and succeed with a high probability with most coin-flip results.

6 Randomization, BPP, and RP

There are a number of predicate classes which incorporate randomization into their respective characteristic functions. The first such predicate class is BPP (or bounded probabilistic polynomial).

Definition: Let $c \in (0, 1/2)$. A predicate L is in the class $BPP(c)$ provided that there is a polynomial time function f and a polynomial $l(n)$ such that for all $x \in \Sigma^*$,

$$x \in L \quad \Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] > \frac{1}{2} + c \quad \text{for random } Y\text{, and}$$
$$x \notin L \quad \Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] < \frac{1}{2} - c.$$

Definition: In this class, we will take BPP to refer to the class $BPP(1/6)$; i.e., where $c = 1/6$.

In defining BPP this way, there is a 2/3 probability of the function returning 1 if the element x is in the predicate, and a 1/3 probability of the function returning 1 if the element x is not in the predicate.

In comparing predicate classes, it is often useful to define characteristic functions in terms of other characteristic functions, for which the following definition is useful.

Definition: A *k-ary predicate* is a subset of $(\Sigma^*)^k$.

Homework Exercise 1. Show that

$$BPP\left(\frac{1}{n^c}\right) = BPP(\delta) = BPP\left(2^{-n^c}\right),$$

where c and δ are any constants, and $n = |x|$. (*Hint:* run the function $f(x, Y)$ on random inputs Y repeatedly, use majority vote to decide the answer, and use Chernoff bounds to obtain the probability the answer is correct.)

The next predicate class is known as RP (or randomized polynomial).

Definition: Let $c \in (0, 1)$. A predicate L is in the class $RP(c)$ provided there is a polynomial-time function f, and a polynomial $l(n)$ such that for all $x \in \Sigma^*$,

$$\begin{aligned} x \in L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] > c, \quad \text{and} \\ x \notin L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] = 0. \end{aligned}$$

Definition: In this class, we take RP to be $RP(1/2)$; i.e., where $c = 1/2$.

Theorem. Let $n = |x|$ (we are interested primarily in the asymptotic behavior of n when comparing it to constants). The three classes $RP(1/2)$, $RP(1/n)$, and $RP(1 - 2^{-n})$ are equal.

Proof. First we will show $RP(1/n) = RP(1/2)$. The reverse containment is easy, by noticing that $1/2 > 1/n$; if $x \in L$ and $f(x, Y) = 1$ with probability greater than 1/2, the probability is also certainly greater than $1/n$.

For the forward containment, let f be a polynomial-time function, and let $l(n)$ be a polynomial such that

$$\begin{aligned} x \in L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] > \frac{1}{n}, \quad \text{and} \\ x \notin L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] = 0. \end{aligned}$$

Define the function g as follows:

$$\begin{aligned} g : \Sigma^* \times (\Sigma^*)^n \quad &\rightarrow \quad \{0, 1\}, \quad \text{where} \\ g(x, Y_1, \ldots, Y_n) \quad &= \quad max(f(x, Y_1), \ldots, f(x, Y_n)), \end{aligned}$$

and $Y_1, \ldots, Y_n$ are random values. The runtime of g is approximately a factor of n times the runtime of f, and so g is still a polynomial-time function of n. If $x \notin L$, then $g(x, Y_1, \ldots, Y_n) = 0$ with probability 1. If $x \in L$, then $g(x, Y_1, \ldots, Y_n) = 0$ only if $f(x, Y_i) = 0$ for all $i \in \{1, \ldots, n\}$. This occurs with probability $(1 - 1/n)^n$, and so $g(x, Y_1, \ldots, Y_n) = 1$ with probability

$$1 - \left(1 - \frac{1}{n}\right)^n \approx 1 - \frac{1}{e} > \frac{1}{2}.$$

Therefore $RP(1/n) = RP(1/2)$.

Showing $RP(1/2) = RP(1-2^{-n})$ uses the same proof construct, with a slight variation of the computation of the probability of $g = 1$; in particular, considering $RP(1/2)$, the probability that $f(x, Y_i) = 0$ for all $i \in \{1, \ldots, n\}$ is less than $(1-1/2)^n$, and so $g(x, Y_1, \ldots, Y_n) = 1$ with probability greater than

$$1 - \left(1 - \frac{1}{2}\right) = 1 - 2^{-n}.$$

7 Infeasibility and PP

There are certainly a number of predicate classes for which decidability of membership is deemed infeasible. Predicate classes whose characteristic functions are exponential-time are deemed infeasible. As there have been no polynomial-time or otherwise feasible characteristic functions demonstrated for members of the class NP, this class is presumed infeasible. There is another similar predicate class which is presumed infeasible, called PP (or probabilistic polynomial).

Definition: A predicate L is in PP provided that there exists a polynomial-time function f and a polynomial $l(n)$ such that for all $x \in \Sigma^*$,

$$\begin{aligned} x \in L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] \geq \frac{1}{2}, \quad \text{and} \\ x \notin L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] < \frac{1}{2}. \end{aligned}$$

Definition of this predicate class provides an alternate method of analyzing the question of $P = NP$, to which the next exercise alludes.

Homework Exercise 2. Prove that if $P = PP$, then $P = NP$. It may be useful to work with the following definition of the class NP.

Definition: A predicate L is in NP provided that there exists a polynomial-time function f and a polynomial $l(n)$ such that for all $x \in \Sigma^*$,

$$\begin{aligned} x \in L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] > 0, \quad \text{and} \\ x \notin L \quad &\Leftrightarrow \quad Pr_{Y \in \Sigma^{l(|x|)}}[f(x, Y) = 1] = 0. \end{aligned}$$

Homework exercise 2 shows the importance of the fact that BPP and RP are defined by using a fixed constant c strictly greater than zero.

Math 267a - Foundations of Cryptography

Lecture #3: 15 January 1997

Lecturer: Sam Buss

Scribe Notes by: Jason Ribando

8 Last Time

Deemed feasible: P, BPP, RP
Presumed Infeasible: NP, PP

9 P/poly, "Polynomial size circuit"

The class of P/poly functions is also deemed feasible.

Definition: Let $f : \Sigma^* \to \{0,1\}^*$. We say $f \in P/poly$ provided there is a polynomial time algorithm $A : \Sigma^* \times \Sigma^* \to \{0,1\}$ and there is a function $y : \mathbf{N} \to \Sigma^*$ such that for all n, $|y(n)| = n^{O(n)}$ and such that for all x, $f(x) = A(x, y(|x|)$.

One can think of $y(n)$ as encoding a circuit that computes f as inputs of length n.

Definition: (Informal) A *circuit* has gates $\wedge$, $\vee$, $\neg$, and n inputs, one for each bit of length n.

Homework #3:
Show that BPP and $RP \subseteq P/$poly.
Hint: Show that for a given n (where n is the length of inputs) there is a small set $\mathcal{Y}$ of values for Y such that

$$\Pr_{Y \in \mathcal{Y}}[f(x, Y) = 1]$$

correctly indicates the value of

$$\Pr_{Y \in \{0,1\}^n}[f(x, Y) = 1].$$

Then let $y(n)$ be a string encoding the appropriate $\mathcal{Y}$.

10 Function Ensembles

If $f : \Sigma^* \to \Sigma^*$ we often assume that $|f(x)|$ is determined by $|x|$, ie. we have a function $l(n)$ such that $|f(x)| = l(n)$ if $|x| = n$.

Definition: We call such a function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ a *function ensemble.*

Definition: A *predicate* is a function ensemble $f : \{0,1\}^n \to \{0,1\}$.

Definition: A *probability distiribution ensemble* $\mathcal{D}$ is a family $\mathcal{D} = \{\mathcal{D}_n\}_{n\geq 1}$ such that $\mathcal{D}_n$ is is a probability distribution on $\{0,1\}^n$.

We write $X \in_{\mathcal{U}} \{0,1\}^n$, and if $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ then this induces a random variable $f(X) \in_{\mathcal{D}} \{0,1\}^{l(n)}$. Here, $\{\mathcal{U}_n\}_{n\geq 1} = \mathcal{U}$ is the uniform distribution.

11 Pseudorandom Number Generators

Definition: Let $g : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a polynomial-time function ensemble with $l(n) > n$. Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$. The function g is a *pseudorandom number generator* provided: for any "adversary" A such that $A : \{0,1\}^{l(n)} \to \{0,1\}$ and A is polynomial-time computable, the *success probability* of A

$$\delta_A(n) = |\Pr_X[A(g(x)) = 1] - \Pr_Y[A(Y) = 1|,$$

is negligible.

Definition: The function $h(n)$ is *negligible* provided $h(n) = n^{o(1)}$, i.e., for all polynomials $p(n)$, $h(n) < 1/p(n)$ for n sufficiently large.

This definition needs tweaking:

An adversary is allowed to be a randomized polynomial-time function, i.e., $A(Y,Z)$ with Z truly random: $Z \in_{\mathcal{U}} \{0,1\}^{r(n)}$. Then change the definition so the success probability, now defined to equal

$$\delta_A(n) = |\Pr_{X,Z}[A(f(X),Z) = 1] - \Pr_{Y,Z}[A(Y,Z) = 1]|$$

is negligible. One can think of Z as random coin flips (or bits) used by A.

Definition: A function g is a $S(n)$-*secure pseudorandom number generator* provided that for every adversary as above (though not necessarily polynomial-time computable) we have that $S(n) \leq T(n)/\delta_A(n)$, where $T(n)$ is the worst case running time of A on inputs of length $l(n)$. We call $T(n)/\delta_A(n)$ the *time-success ratio of* A.

We'd like to have $S(n) = 2^{\sqrt{n}}$, for instance.

Homework #4: (Easy)

If $P = NP$, pseudorandom number generators do not exist.

Conjectured pseudorandom number generator:

Select p and q to be n-bit primes congruent to 3 mod 4.
Let $z = pq$.
Choose x relatively prime to p and q.
Let $x_0 = x$ and $x_{i+1} = x_i^2 \bmod z$.
Let a_i be the lower order bit of x_i.
Output $a_1, a_2, \ldots, a_{4n} \in \{0,1\}^{4n}$.

The function $(p, q, x) \rightarrow a_1 a_2 \cdots a_{4n}$ is conjectured to be a pseudorandom number generator, even if pq is publicly known to the adversary.

Math 267a - Foundations of Cryptography

Lecture #4: 17 January 1997

Lecturer: Sam Buss

Scribe Notes by: Chris Pollett

12 One way functions

Informal idea: A one way functions is a function

$$f : \Sigma^n \longrightarrow \Sigma^{p(n)}$$

which is easy to compute (i.e., p-time); however, an adversary A given $f(x)$ has a "hard" time finding an x' such that $f(x') = f(x)$. (x' may equal x.)

Example 12.1 Product: Let $x, y > 1$ be n-bit integers Define

$$f(x, y) = x \cdot y.$$

When x, y are n-bit primes, it is believed that finding x, y from $x \cdot y$ is computationally difficult. Since x is prime with probability $\approx 1/n$ (By the prime number theorem $\pi(x) \to x/\log x$ and $n \approx \log x$.) both x, y are prime with probability $\approx 1/n^2$. Product is a so-called "weak" one-way function which roughly means it has a certain fraction of computationally difficult instances.

Example 12.2 Discrete Log: Let p be an n-bit prime. Let g be a generator of $\mathbb{Z}_p^*$. Define

$$f : (p, g, x) \mapsto (p, g, g^x \bmod p).$$

Example 12.3 Root Extraction: (RSA function) Let p, q be n-bit primes. Let e be relatively prime to $(p-1)(q-1)$. So the order of $\mathbb{Z}_{pq}^*$ is $(p-1)(q-1)$. Define

$$f : (p, q, e, x) \mapsto (p \cdot q, e, x^e \bmod pq).$$

To invert $(p \cdot q, e, y)$ need to find $\sqrt[e]{y}$ in $\mathbb{Z}_{pq}^*$. The function $f_{p,q,e}(x) = x^e \bmod p \cdot q$ is one-to-one in $\mathbb{Z}_{pq}^*$.

Example 12.4 Square Root: Even the $e = 2$ case of the last example is believed to be one way. This case is a four-to-one function as e is not relatively prime to $(p-1)(q-1)$.

Each of the above schemes relies on it being feasible to raise a number to a power modulo another number. We now spend a moment to justify this.

Theorem 12.5 *Computing $a^b \bmod z$ where $a, b < z$ can be done in time polynomial in the length of a, b, z.*

Proof: The recursive algorithm is as follows:

If b is even compute $a^b \bmod z$ by squaring $a^{(b/2)} \bmod z$ modulo z.

If b is odd then compute $a^b \bmod z$ by first squaring $a^{(b/2)} \bmod z$ then multiplying by $a \bmod z$ modulo z.

This gives a polynomial time algorithm as multiplication $\bmod z$ is polynomial time and it takes $O(|b|) = O(n)$ squarings or squaring with a multiplication to compute $a^b \bmod z$. □

Now we consider some more examples of one-way functions. This time trying to avoid number theory.

Example 12.6 Subset Sum: Let $b_1, \ldots, b_n$ be n-bit integers chosen at random and let $a \subseteq \{1, \ldots, n\}$ be coded by an element in $\{0,1\}^n$. Define

$$f : (b_1, \ldots, b_n, a) \mapsto (\sum_{i \in a} b_i,\ b_1, \ldots, b_n).$$

It conjectured this is hard to invert even though $b_1, \ldots, b_n$ are known.

Example 12.7 Linear Codes: A (k, n, d) linear code is a $k \times n$ matric over $\mathbb{Z}_2$ such that for all $x \neq 0$, $x \in \{0,1\}^k$ the vector xC has at least d ones. The idea is if $x \neq y$ then $xC - yC$ has at least d ones so we can correct $< d/2$ errors if we use a linear code as an error correcting code. To use a linear code as a one way function define

$$f : (C, x, e) \mapsto (C, x, xC + e)$$

where e has $< d/2$ ones. This is conjectured to be one way.

Now that we have the intuitive idea of one-way function and these examples under our belt, let's try to give a precise definition.

Definition 12.8 *Let $f : \{0,1\} \longrightarrow \{0,1\}^{p(n)}$ be a polynomial time computable function ensemble. Then f is a* one-way function *provided for all adversaries A which are random p-time, the success probability of A,*

$$\delta_A(n) = Pr_{x \in \{0,1\}^n}[f(A(f(x))) = f(x)],$$

is negligible.

Definition 12.9 *A function f is an $S(n)$-*secure one-way function *if any adversary (not necessarily p-time) has time-success ratio $T_A(n)/\delta_A(n) \geq S(n)$ where $T_A(n)$ is the worst case run-time of A on inputs of length n.*

Definition 12.10 *A function is a* one-way permutation *if $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ is a one-way function and is one-to-one.*

13 One-way functions with public input

Consider a function $f : \{0,1\}^n \times \{0,1\}^{q(n)} \longrightarrow \{0,1\}^{p(n)}$ where the $\{0,1\}^n$ are supposed to be vectors of privately known bits and the $\{0,1\}^{q(n)}$ are supposed to be vectors of publicly known bits. We define the success probability of an adversary A trying compute an inverse to f as

$$\delta_A(n) = Pr_{X \in \{0,1\}^n, Y \in \{0,1\}^{q(n)}}[f(A(f(x,Y),Y),Y) = f(X,Y)].$$

Given δ_A we can modify the definitions above to define a one-way function with public input, an $S(n)$-secure one way function with public input, and a one-way permutation with public input.

Math 267a - Foundations of Cryptography

Lecture #5: 22 January 1997

Lecturer: Sam Buss

Scribe Notes by: Dell Kronewitter

14 Pseudo-random number generators and one way functions

Theorem 14.1 *Any pseudo-random number generator is a one way function.*

Proof:
First we'll show the theory for the case of a deterministic adversary.
Intuistic idea of proof:

Suppose f is a pseudo-random number generator, and yet f is not a one way function. Suppose our deterministic adversary against one way function-ness, A, has success probability 1. (i.e. We have $f(A(x)) = (x)$ for all x.) Next, we convert our one-way adversary, A, into a pseudo-random adversary B. We define the action of B as follows :

$$B(y) = \begin{cases} 1 & \text{if } f(A(y)) = y \\ 0 & \text{otherwise} \end{cases}$$

Let's do some calculations :

The above definitions mean in particular that: A has run time $T(n)$. A has success probability $\delta_A(n) = 1$. f is in P-time. A has time success ratio $T(n)/1 = T(n)$. B has run time $\approx T(n) + n^{\theta(1)}$ which is without loss of generality $T(n)$. (It is still polynomial.)

Let's analyze B's effectiveness and see if it cuts the mustard.

Let $X \in_{\mathcal{U}} \{0,1\}^n$ (the domain of f)

Let $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$ (the co-domain of f {nicely described superset of the range of $f(X)$ })

Since

$$Pr_{X\in\{0,1\}^n}[f(A(f(X))) = f(X)] = 1$$

We have that

$$Pr_{X\in\{0,1\}^n}[B(f(X)) = 1] = 1$$

and

$$Pr_{Y\in\{0,1\}^{l(n)}}[B(Y) = 1] \leq 2^n/2^{l(n)} = 2^{n-l(n)}$$

So B has success probability $\delta_B(n) = |1 - 2^{n-l(n)}| \geq 1/2$ and the time success ratio of B is $\approx T(n)/(1/2) = 2T(n)$ = twice the time success ratio of A.

This means B meets the criteria for being an adversary which proves that f is not a pseudo-random number generator. So ends the deterministic adversary case.

Proof of Thereom: (Remove deterministic restriction)[1]

Let A be an adversary against f being a one way function. Let A have run time $T(n)$ and success probability $\delta(n)$. Next, we define adversary B' :

First calculate $z = A(y)$.
If $f(z) = y$ then $B' = 1$.
Otherwise, with probability 1/2, B' outputs 1, and, with probability 1/2, B' outputs 0.

Note that the adversary B' first tries to invert f on y; if this fails, it then makes a guess as to whether y is in the range of f.

The runtime of $B' \approx T(n)$. (B' performs A's calculations and then f's.)
To find the success probability of B, we estimate as follows:

$$\begin{aligned} Pr_{X\in\{0,1\}^n}[B'(f(X))=1] &= Pr_{X\in\{0,1\}^n}[B'(f(X))=1] \\ &\geq \delta(n)\cdot 1 + (1-\delta(n))(1/2) = 1/2 + (1/2)\delta(n). \end{aligned}$$

$$\begin{aligned} Pr_{Y\in\{0,1\}^{l(n)}}[B'(Y)=1] &\leq 2^{n-l(n)}(1/2+(1/2)\delta(n)) + (1-2^{n-l(n)})(1/2) \\ &\leq (1/2)(1/2+(1/2)\delta(n)) + 1/2 \\ &= 1/2 + (1/4)\delta(n). \end{aligned}$$

(Recall that we have chosen $l(n) \geq n+1$)

Now we have that the success probability of B' is greater than $(1/4)\delta(n)$. So the time-success ratio of $B' \leq T(n)/(1/4)\delta(n) = 4$(time-success ratio of A) □

A weaker converse holds whose proof will be presented later.

Theorem 14.2 *If there exists a one way function then there exists a pseudo-random number generator.*

Remark 14.3 Do not miss the significance of the above theorem. The existence of pseudo-random number generators might at first seem questionable while one way functions are quite believable. Examples of one way functions are "easy" to find. Examples of pseudo-random number generators are much harder to find.

Definition 14.4 *Here let us recall our definition of* NP*: Let $f(x,y)$ be P-time computable, then $A \in NP$ can be defined as $A(x)$ if and only if there exists a $y \in \{0,1\}^{p(|x|)}$ such that $f(x,y) = 1$*

[1] This proof is little more complicated than it needs to be. [S.B.]

Definition 14.5 *An NP* search problem for $f(x, y)$, p(n) *is solved for any g(x) such that for all* x

1. *If there is a $y \in \{0,1\}^{p(|x|)}$ such that $f(x, y) = 1$ then $g(x)$ is such a y.*
2. *Otherwise $g(x) = 0$*

15 Weak one way functions

Definition 15.1 *Let $f : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$ and suppose $w(n)$ is a non-negligible function. (i.e. $w(n) \geq 1/n^c$ for some $c > 0$) Then f is an* $S(n)$-secure $w(n)$-weak one way function *provided that for any adversary $A : \{0,1\}^{l(n)} \longrightarrow \{0,1\}^n$ either A has runtime $T_A(n) < S(n)$ or A has success probability $\delta_A(n) \leq 1 - w(n)$ (i.e. A has probability of failure $\geq w(n)$) We can say there is a non-negligible chance of inverting the function.*

Definition 15.2 *A function f is a* weak one way function *provided there is a non-negligible $\epsilon(n)$ such that every polynomial time adversary has success probability $< 1 - \epsilon(n)$*

Exercise 5) Prove that if P = NP then one-way functions don't exist. (Hint: Consider Exercise 6)

Exercise 6) Prove that if P = NP then NP search problems are solvable in polynomial time. (Hint: Let y_0 = least such y solving search problem (if there is one). Solve NP problems to determine the $l(n)$ bits in y_0 one at a time.)

Math 267a - Foundations of Cryptography

Lecture #6: 24 January 1997

Lecturer: Sam Buss

Scribe Notes by: Mike Mastropietro

16 Converting Weak One Way Functions to One Way Functions

Theorem 16.1 *If there is a weak one way function f, then there is a one way function g.*

Proof:
Construction:

Let f be an $R_f(n)$ secure $w(n)$ weak one way function. [*Note*: $w(n) \approx \frac{1}{n^c}$]

$$f : \{0,1\}^n \to \{0,1\}^{l(n)}$$

Define g as follows: let $N = 2n/w(n)$

$$g : \{0,1\}^{n \times N} \to \{0,1\}^{l(n) \times N}$$

By interpreting $y \in \{0,1\}^{n \times N}$ as $y = y_1 y_2 \cdots y_N$, $y_i \in \{0,1\}^n$ then:

$$g(y) = f(y_1) f(y_2) \cdots f(y_N)$$

We'll show that g is a R_g secure one way function, where g has inputs of size nN, and view R_g as a function of nN. (*Note:* nN is called the *security parameter*, that is, the number of bits to be kept secret.)

We'll have

$$R_f(n) \le n^{O(1)} R_g(n^{O(1)})$$

If $R_f(n)$ is not bounded by a polynomial, (i.e. f is a weak one way function), then neither is $R_g(n)$, so that g is a one way function.

Idea of proof:

Suppose we have an adversary A for g. From A, we shall construct an adversary S^A for f.

The input into S^A is $z \in \{0,1\}^{l(n)}$ The goal of S^A is to find a value $x \in \{0,1\}^n$ such that $f(x) = z$.

The Algorithm for S^A is as follows:

Repeat sufficiently many times: (number to be specified later)

1. Choose $i \in_U \{1, 2, \cdots, N\}$
2. Choose $x_1, x_2, \cdots, x_{i\text{-}1}, x_{i+1}, \cdots, x_N \in \{0,1\}^n$
3. Form $\alpha = f(x_1) \cdots f(x_{i\text{-}1}) z f(x_{i+1}) \cdots f(x_N)$
4. Set $u = A(\alpha)$, and view u as $u = u_1 u_2 \cdots u_N$
5. If $f(u_i) = x$ output u_i and halt.
   Otherwise, loop.

To prove the Theorem, we still need to analyze the runtime and success probability of S^A.

Probability Analysis:

Fix $n >> 0$. Let $\mathcal{F} = \{0,1\}^n, \mathcal{G} = \{0,1\}^{n \times N}$ Define a bipartite multi-graph, H on $\mathcal{F} \cup \mathcal{G}$, where the edges are $\{(x,y) | x \in \mathcal{F}, y \in \mathcal{G}$ and $x = y_i$, where $y = y_1 \cdots y_N\}$ An edge (x, y) is present once for each occurrence of x in $y = y_1 \cdots y_N$.

Definition 16.2 *For $x \in \mathcal{F}$, define $adj(x)$, the* adjacency set of x by

$$adj(x) = \{y | (x,y) \in H\}$$

This is a multiset, as is $adj(y) = \{x | (x,y) \in H\}$. Further $|adj(y)| = N$ and $|adj(x)| = N \cdot 2^{n(N-1)}$

Definition 16.3 *Let $X \in_U \mathcal{F}$, $Y \in_U \mathcal{G}$, and $\epsilon, \delta \geq 0$. We say H has (ϵ, δ)* forward expansion if for every $F \subseteq \mathcal{F}$, if $\Pr_X[X \in F] = \frac{|F|}{|\mathcal{F}|} \geq \epsilon$ then $\Pr_Y[\exists x \in F | Y \in adj(x)] \geq 1 - \delta$

Intuition for definition: Suppose A_f is a deterministic adversary for f. We can build and adversary A_g for g as follows:
Given input, $y_1, \cdots, y_N$ for g compute $A_f(y_1) \cdots A_f(y_N)$. If all these succeed in finding $z_i = A_f(y_i)$ such that $f(z_i) = y_i$, then g outputs $z_1, \cdots, z_N$.

Claim 1 *If H has (ϵ, δ) forward expansion, and A_f has failure rate ϵ, then A_g has failure rate $\geq 1 - \delta$.*

Proof of Claim: Let $F = \{x \in \mathcal{F} | A_f(f(x))$ fails$\}$ Then A_g succeeds only if A_f succeeds for all $y_i = f(z_i)$, i.e., none of the $z_1, \cdots, z_N$ are in F. This happens when $z_1, \cdots, z_N$ is not adjacent to any element of F. (This happens with probability $\leq \delta$.)

Lemma 16.4 *For any $0 < \epsilon < 1$, H has $(\epsilon, (1-\epsilon)^N)$ forward expansion.*

Proof: $\Pr_Y[Y \in adj(x), \text{some } x \in F] = \Pr_{Y_1 \cdots Y_N \in F}[Y_i \in F, \text{some } i] = 1 - (1-\epsilon)^N = 1 - \delta$ □

Corollary 16.5 *H has $\left(\frac{w(n)}{2}, e^{-n}\right)$ forward expansion.*

Proof: H has $\left(\frac{w(n)}{2}, \left(1-\frac{w(n)}{2}\right)^N\right)$ forward expansion, but

$$\left(1-\frac{w(n)}{2}\right)^N = \left(1-\frac{w(n)}{2}\right)^{\frac{2n}{w(n)}} = \left(\left(1-\frac{w(n)}{2}\right)^{\frac{2}{w(n)}}\right)^n \leq \left(\frac{1}{e}\right)^n = e^{-n}$$

$\square$

NEXT TIME: "Reverse Expansion" The proof of the Theorem will continue next time.

Math 267a - Foundations of Cryptography

Lecture #7: 27 January 1997

Lecturer: Sam Buss

Scribe Notes by: Tyler McIntosh

17 Reverse expansion

Recall from Lecture 6: H is a bipartite graph on $\mathcal{F} \cup \mathcal{G}$ with

$$\mathcal{F} = \{0,1\}^n\text{- inputs to } f$$

$$\mathcal{G} = \{0,1\}^{n \times N}\text{- inputs to } g$$

and edges (x, y) such that x appears in y

$$x \in \mathcal{F}$$

$$y = y_1 y_2 \ldots y_N \in \mathcal{G}$$

H has forward expansion $(w(n)/2, e^{-n})$.

Definition 17.1 *H has $(\epsilon, \delta, \gamma)$-*reverse expansion *if and only if for all $G \leq \mathcal{G}$ such that $Pr_Y[Y \in G] = \frac{|G|}{|\mathcal{G}|} > \delta + \gamma$ there is a set $F \in \mathcal{F}$ such that:*

(1) $Pr_X[X \in F] \geq 1 - \epsilon$

(2) For $x \in F, Pr_{Y(x)}[Y(x) \in G] \geq \gamma/N$

where $Y(x) \in_{\mathcal{U}} adj(x)$.

Lemma 17.2 *If H has (ϵ, δ) forward expansion, and if $\gamma > 0$ then H has $(\epsilon, \delta, \gamma)$ reverse expansion.*

Proof:
Let $\frac{|G|}{|\mathcal{G}|} \geq \delta + \gamma$. Let $\overline{F} = \{x : Pr_{Y(x)}[Y(x) \in G] < \gamma/N\}$. We need to show $\frac{|\overline{F}|}{|\mathcal{F}|} < \epsilon$. Let us assume it is not. By the (ϵ, δ)-forward expansion definition, the set

$$G' = \{y : \text{for some } x \in \overline{F},\ y \text{ is adjacent to } x\}$$

satisfies $\frac{|G'|}{|\mathcal{G}|} \geq 1 - \delta$.
Let $G'' = G \cap G'$ so $|G''| \geq \gamma|\mathcal{G}| \geq \gamma|G'|$.

Recall: Each node in $\mathcal{G}$ has N neighbors in $\mathcal{F}$.
Each y' in G' has at most N edges to $\overline{F}$.
Each y' in G'' has at least one edge to $\overline{F}$ (since $G' \supseteq G''$).
So at least γ/N fraction of all edges from $\overline{F}$ go to G''.
Some $x \in \overline{F}$ has at least γ/N edges to G'' (and hence to G).
However, this contradicts the definition of $\overline{F}$.

□

18 Let's prove Theorem from Lecture #6

Proof:

Let's assume A is our adversary for g. For the moment, we assume A is deterministic. The worst case runtime for A is $T(n)$ (on inputs of length nN). Success probability of A is $\delta(n)$. Time-success ratio of A is $R_g(n) = T(n)/\delta(n)$.

Without loss of generality: $\delta(n) > 2e^{-n}$

[Note: This can be assumed without loss of generality since g has adversary with runtime $\approx 2^n$ and success probability 1 (which just used a brute-force search). This has time-success ratio $n^{O(1)}2^n/1 < \frac{e^n}{2}$]

Let $G = \{y \in \mathcal{G} : g(A(y)) = y\}$,

$$\frac{|G|}{|\mathcal{G}|} = \delta(n) = \frac{\delta(n)}{2} + \frac{\delta(n)}{2}$$

H has $(\frac{w(n)}{2}, \frac{\delta(n)}{2}, \frac{\delta(n)}{2})$-reverse expansion since it has $(\frac{w(n)}{2}, e^{-n})$-forward expansion.

Let $F = \{x \in \mathcal{F} : Pr_{Y(x)}[Y(x) \in G] \geq \frac{\delta(n)}{2N}\}$

$Pr_X[X \in F] \geq 1 - \frac{w(n)}{2}$

Recall the algorithm from Lecture#6:

The Algorithm for S^A is as follows:

Input: $z = f(x)$ where x is unknown
Repeat $\frac{2nN}{\delta(n)}$ times:

1. Choose $i \in_U \{1, 2, \cdots, N\}$
2. Choose $x_1, x_2, \cdots, x_{i\text{-}1}, x_{i+1}, \cdots, x_N \in \{0,1\}^n = \mathcal{F}$
3. Form $\alpha = f(x_1)\cdots f(x_{i\text{-}1})zf(x_{i+1})\cdots f(x_N)$
4. Set $u = A(\alpha)$, and view u as $u = u_1u_2\cdots u_N$
5. If $f(u_i) = z$ output u_i and halt.
   Otherwise, continue looping.

Note that, in the above algorithm, α is chosen at random uniformly $g(adj(x))$

The probability that one iteration of the loop succeeds is $\geq \delta(n)/2N$ if $\alpha \in G$ provided $x \in F$.

For $x \in F$, all $2nN/w(n)$ loops fail with probability $\leq (1 - \frac{\delta(n)}{2N})^{\frac{2nN}{\delta(n)}} \leq e^{-n}$. Therefore:

$$Pr_X[S^A(f(X)) \text{ fails}] \leq Pr[X \notin F] + e^{-n} \leq \frac{w(n)}{2} + e^{-n} < w(n)$$

So S^A succeeds with probability $\geq 1 - w(n)$.
The runtime of S^A is $\frac{2nN}{\delta(n)}(T(n) + n^{O(1)}) \approx \frac{2nN}{\delta(n)}(T(n))$.
Therefore, the time-success ratio of S^A:

$$\begin{aligned} S^A &\approx (2n/\delta(n))(2n/w(n))(T(n)/(1-w(n))) \\ &= n^{O(1)}T(n)/\delta(n) \\ &= n^{O(1)}R_g(nN) \end{aligned}$$

Thus,

$$R_f(n) \leq n^{O(1)}R_g(n^{O(1)})$$

That completes the proof of the Theorem for the case where f does not use any public input and where the adversary is deterministic. The next two observations show that the above proof also applies to the general case.

Observation 1: The same proof works if there's a additional "public" input. For public input π where $f : x, \pi \mapsto y, \pi$:

$$f : \{0,1\}^n \times \{0,1\}^{p(n)} \to \{0,1\}^{l(n)} \times \{0,1\}^{p(n)}$$

$$g : \{0,1\}^{n\times N} \times \{0,1\}^{p(n)\times N} \to \{0,1\}^{l(n)\times N} \times \{0,1\}^{p(n)\times N}$$

$$g : y_1\pi_1, y_2\pi_2, \ldots, y_n\pi_n \mapsto f(y_1\pi_1), f(y_2\pi_2), \ldots, f(y_n\pi_n)$$

$$\mathcal{F} = \{0,1\}^n \times \{0,1\}^{p(n)}$$

$$\mathcal{G} = \{0,1\}^{n\times N} \times \{0,1\}^{p(n)\times N}$$

H is now a bipartitie graph on these larger sets F and G, but the rest of the proof is the same.

Observation 2: If A uses randomization, then the random string A uses can be viewed as a public input which f just ignores.

$$Pr_{X,\pi}[f(A(F(x,\pi),\pi),\pi) = f(x,\pi)]$$

The only mathematical difference between randomization chosen by A and public inputs chosen by f is that f is allowed to use the public inputs, but not allowed use random bits generated by A (and in any event, A usually chooses its random bits after the value of f has been computed). The operational difference, as used in a cryptographic application, is that f chooses the public input at random in the knowledge that the adversary A will fail for nearly every public input; therefore, f doesn't want A to be able to influence the choice of public input. Likewise, A doesn't want the encrypter f to be able influence A's random choices. $\square$

Math 267a - Foundations of Cryptography

Lecture #8: 29 January 1997

Lecturer: Sam Buss

Scribe Notes by: Jennifer Wagner

19 (Weak) One-way permutations

Definition 19.1 *A (weak) one-way function $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ is a* (weak) one-way permutation *provided that it is one-one and onto.*

The previous theorem started with a $w(n)$-weak one-way function $f : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$ with security parameter $s_f(n) = n$ and constructed a one-way function $g : \{0,1\}^{n*N} \longrightarrow \{0,1\}^{l(n)*N}$, where $N = \frac{2n}{w(n)}$ with security parameter $s_g(n) = N = n^{\mathcal{O}(1)}$. This significantly increases the number of bits that must be kept secret. The following construction allows us to create a one-way permutation g from a weak one-way function f, maintaining the same number of secret bits. In Luby's terminology, this is a "linear preserving reduction," the best kind.

<u>Construction:</u> Let $f : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$ be a $R_f(n)$-secure $w(n)$-weak one-way function. Let $N = \frac{2n}{w(n)}(= n^{\mathcal{O}(1)})$. Let $\pi = \pi_1\pi_2...\pi_N$ be public information with each $\pi_i \in \{0,1\}^n$. Then define $g : \{0,1\}^n \times \{0,1\}^{n*N} \longrightarrow \{0,1\}^n \times \{0,1\}^{n*N}$ by $g(x,\pi) = (y_{N+1}, \pi)$ where y_{N+1} is found recursively: Set $y_1 = x$. Then let $y_{i+1} = \pi_i \oplus f(y_i)$ for $i = 1,...,N$. The following illustrates this process.

$$x = y_1 \rightarrow f(y_1) \rightarrow y_2 = f(y_1)\oplus\pi_1 \rightarrow f(y_2) \rightarrow y_3 = f(y_2)\oplus\pi_2 \rightarrow ... \rightarrow f(y_N) \rightarrow y_{N+1} = f(y_N)\oplus\pi_N$$

Theorem 19.2 *In the above construction, g is an $R_g(n)$-secure one-way permutation such that $R_f(n) \leq n^{\mathcal{O}(1)} R_g(n)$. The security parameter of g is $s_g(n) = n$.*

Proof: (Sketch) Let A be an adversary for g with worst-case runtime $T(n)$ and success probability $\delta(n)$. We form an adversary S^A for f according to the following algorithm.

Algorithm: The input to S^A is $z = f(x)$.

Repeat the following loop $\frac{2nN}{\delta(n)}$ times:
Choose $i \in_{\mathcal{U}} \{2, ..., N+1\}$.
Choose $\pi \in_{\mathcal{U}} \{0,1\}^{n*N}$.
Let $y_i = z \oplus \pi_{i-1}$.
Set $y_{j+1} = \pi_j \oplus f(y_j)$ for $j = i...N$.
Calculate $\alpha = A(y_{N+1}, \pi)$. Note that we hope $\alpha = (y_1, \pi)$.
Set $y_{j+1} = \pi_j \oplus f(y_j)$ for $j = 1, ..., i-2$.
If $f(y_{i-1}) = z$, then output y_{i-1}.
Otherwise continue the loop.

This algorithm inserts the input z as $f(y_{i-1})$, then continues the original construction. It then uses A to determine y_1 and return to the beginning of the construction, continuing until y_{i-1} is found. If $f(y_{i-1}) = z$, the algorithm was successful.

$$z \to y_i = z \oplus \pi_{i-1} \to f(y_i) \to y_{i+1} = f(y_i) \oplus \pi_i \to ... \to f(y_N) \to y_{N+1} = f(y_N) \oplus \pi_N$$

$$\to y_1 = A(y_{N+1}) \to f(y_1) \to y_2 = f(y_1) \oplus \pi_1 \to ... \to f(y_{i-2}) \to y_{i-1} = f(y_{i-2}) \oplus \pi_{i-2}$$

Analysis: The probability analysis is technically similar to that in the previous proof. The following are a few of the details which are different. Here, the set $\mathcal{F} = \{0,1\}^n \times \{0,1\}^{n*N}$. Since f is a permutation, the values (i, y_i, π) completely determine the sequence $(y_1, ..., y_{N+1}, \pi)$. So we can let these sequences be the elements of $\mathcal{G}$, or equivalently, let $\mathcal{G} = \{\langle y_1, \pi \rangle : y_1 \in \{0,1\}^n\}$. Then the bipartite graph H has edges $(y_i, \langle y_1, \pi \rangle)$ where $i \leq N$ and y_i appears in the sequence for (y_1, π). Note that all elements of $\mathcal{G}$ have degree N. If we knew what x was, the algorithm for S^A would then be equivalent to the following.

The input is $z = f(x)$.
Choose $\beta \in_{\mathcal{U}} adj(x)$.
Set $\alpha = A(\beta)$.
If $A(\beta)$ succeeded, then output $f^{-1}(z) = x$.

After this, all of the estimates on the success probability of S^A are the same as in the last proof.
□

One problem with the above construction is that the number of public bits is significantly increased. This could cause problems if the adversary is allowed to pick the public information. Luby's book contains other constructions which linearly preserve the number of public bits as well as the number of private ones.

20 Square Root Extraction and Nontrivial Factoring Problems

Square Root Extraction Problem: Given z and x with $2 \leq x \leq z-1$, x relatively prime to z, and $x = a^2 \bmod z$ for some a, find a value b such that $x = b^2 \bmod z$. An adversary for the square root extraction problem has success rate $\delta(n)$ if for all $z > 2$,

$$\text{Prob}_X[(A(X,z))^2 = X \bmod z] \geq \delta(n)$$

where $X \in_{\mathcal{U}} \{u : 2 \leq u \leq z,\ \gcd(u,z) = 1\}$.

<u>Nontrivial Factor Problem:</u> Given a (composite) number z, find a nontrivial factor.

Theorem 20.1 *If the square root extraction problem has an adversary with runtime $T(n)$ and success rate $\delta(n)$, then the nontrivial factor problem has an adversary with runtime $\frac{T(n)}{\delta(n)}$ and success rate $\geq \frac{1}{2}$. In fact, the success rate can be made arbitrarily close to 1 with runtime increasing by a constant factor.*

Corollary 20.2 *If factoring is "hard," then finding square roots is "hard."*

The proof of the theorem is given in the next lecture.

Math 267a - Foundations of Cryptography

Lecture #9: 31 January 1997

Lecturer: Sam Buss

Scribe Notes by: Preeti K. Mehta

21 A Little Bit of Number Theory

We were talking last time about reducing the problem of factoring to the problem of finding square roots mod n. We would like to restrict our attention to z's which are not powers of primes. The case of $z = pq$, a product of two primes will be the most important application, but we want our theorems to apply to all composites z. A few facts before we get back to where we were last time:

Fact 1: There exists a polynomial-time algorithm which on input z finds the largest m such that z is a power of m.

Proof: Evaluate $\sqrt[2]{z}, \sqrt[3]{z}, \ldots, \sqrt[k]{z}$, where k is the greatest integer $\le \log_2 z$. If $z = m^k$ then $k \le \ log_2 z$. So we can assume that z is not a prime power because otherwise if $z = m^l$ then we can find m and l in polynomial time.

Fact 2: There exists a polynomial-time algorithm which, given $z = p^l$ (a prime power), and $x = a^2 \mod z$ (x, z relatively prime),finds a square root of x.

Homework #7: Prove this second fact (or read and understand a proof).

So our problem is now reduced to taking square roots modulo a composite (not a prime or a prime power). We will also assume z is odd, since otherwise it is easy to find a nontrivial factor of z, namely 2.

Fact 3: If z is not a prime power and $x \equiv a^2 \bmod z$ and $\gcd(x, z) = 1$, then x has at least four square roots modulo z.

Proof: a and $-a$ ($-a \equiv z - a$) are two square roots of x. Write $z = pq$, with p, q relatively prime, but not necessarily prime themselves. Let $a_p \equiv a \bmod p$ and $a_q \equiv a \bmod q$. By the Chinese Remainder Theorem, there are four values a_1, a_2, a_3, a_4 (including a and $-a$) such that $a_i \bmod p \equiv \pm a_p$ and $a_i \bmod q \equiv \pm a_q$. (Assume here that $p, q > 2$ so that a and $-a$ are distinct modulo p and distinct modulo q.)
Since,

$$a_i^2 \equiv x \bmod p \text{ and } a_i^2 \equiv x \bmod q$$

We have that

$$a_i^2 \equiv x \bmod pq$$

.

One more observation: For z not a prime power, suppose $a^2 \equiv b^2 \bmod z$, $a \not\equiv \pm b \bmod z$. Then $a^2 - b^2 = (a+b)(a-b) \equiv 0 \bmod z$. So, $\gcd(a+b,z) \neq 1$, $\neq z$ and $\gcd(a+b,z)$ divides z. Thus we have a nontrivial factor of z. So to find a factor of z, it suffices to find $a^2 \equiv b^2 \bmod z$ with $a \not\equiv \pm b \bmod z$.

Lemma 21.1 *Suppose $A(x,z)$ is an adversary for finding square roots, and supppose $A(x,z)$ has*

$$\Pr_{\substack{x=a^2 \bmod z \\ \gcd(x,z)=1}} [A(x)^2 \equiv x \bmod z] = \delta_z.$$

Then the following algorithm will find a nontrivial factor of z with probability $\geq \frac{1}{2}$. Its runtime is $O(\frac{\text{runtime of } A}{\delta_z})$. I.e., the success probability of the next algorithm is approximately the success probability of A. Here z is assumed to be odd and not a prime power. Note that this lemma proves the theorem stated in the last class.

Algorithm: S^A is an adversary with input z seeking a nontrivial factor of z.
Loop ($\frac{2}{\delta_z}$ times)
  Choose $a \in_{\mathcal{U}} \{2, \ldots, z-l\}$.
  If $gcd(a,z) \neq 1$, output $gcd(a,z)$. This is a nontrivial factor of z.
  Let $x = a^2 \bmod z$
  Let $a' = A(x)$
  Let $x' = (a')^2 \bmod z$
  If $1 < gcd(a+a',z) < z$, output $\gcd(a+a',z)$.
  Otherwise, continue.

Claim: Each iteration of the loop succeeds with probability $\geq \frac{1}{2}\delta_z$
Proof: Adversary $A(y)$, if successful in finding a square root of x, finds one distinct from $\pm a$ with probability

$$1 - \frac{2}{\text{number of square roots of } x}.$$

To see this, note that an equivalent way to choose a, x and a' is to choose x with probability proportional to the number of square roots it has, and then let $a' = A(x)$ and let a be chosen uniformly from the square roots of x.

The claim states that the success probability of one iteration of the loop is $\geq \frac{1}{2}\delta_z$. Therefore, the success probability of S^A is

$$1 - (1 - \frac{\delta_z}{2})^{\frac{2}{\delta_z}} > 1 - e^{-1} > \frac{1}{2}$$

QED Lemma and the theorem from yesterday.

22 Next-bit Unpredictability

Next-bit unpredictability is another characteristic of pseudorandom number generators.

Definition 22.1 *Let $g : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$ with $l(n) > n$ be a polynomial-time ensemble. Let A be an adversary for predicting next bits. Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $I \in_{\mathcal{U}} \{1, \ldots, l(n)\}$ (X, I random variables).*

$$g(X)_i \textit{ is the } i^{th} \textit{ bit of } g(X) \textit{ where } i = 1, \ldots, l(n)$$

Let $g(X)_{\{1,\ldots,i-1\}}$ *be the first* $i-1$ *bits of* $g(X)$.

Then the success probability of A for next-bit prediction *is*

$$\delta(n) = \Pr_{X,I}[A(I, g(X)_{\{1,\ldots,I-1\}}) = g(x)_I]$$

Definition 22.2 *g is* $S(n)$-secure next-bit unpredictable **iff** *every adversary A has time-success ratio $\geq S(n)$. (We want the next bit to be completely unpredictable based on what we already have.) We say g is* next-bit unpredictable $\iff$ *g is n^c-secure next-bit unpredictable* $\forall c > 0$.

Theorem 22.3 *g is next-bit unpredictable $\iff$ g is a pseudorandom number generator.*

Homework #8: Prove the reverse direction.
Proof: (of the forward direction) Suppose g is not a pseudorandom number generator (PRNG). Let A be an adversary against g as a PRNG. Let A have runtime $T(n)$ and success probability $\delta(n)$. We must construct another adversary S^A that is good at predicting next bits of g. Let Y_i be random variables ($i = 0, \ldots, l(n)$) defined by

$$Y_i = g(X)_{\{1,\ldots,i\}} Z_i \text{ where } Z_i \in_{\mathcal{U}} \{0,1\}^{l(n)-i}$$

I.e., replace the last $l(n) - i$ bits of $g(X)$ with random bits.

$$\begin{aligned}
\text{Let } \delta_i &= \Pr_{Y_i}[A(Y_i) = 1] \\
\text{So, } \delta_o &= \Pr_{Z \in_{\mathcal{U}} \{0,1\}^{l(n)}}[A(Z) = 1] \\
\delta_{l(n)} &= \Pr_X[A(g(X)) = 1]
\end{aligned}$$

So $\delta(n) = \delta_{l(n)} - \delta_0 =$ success probablity of A (how well A distinguishes between output of g and random output).
Define e_i and δ_i^- by

$$\begin{aligned}
e_i &= \overbrace{0\cdots0}^{i-1}1\overbrace{0\cdots0}^{l(n)-i} \\
\delta_i^- &= \Pr_{Y_i}[A(Y_i \oplus e_i) = 1]
\end{aligned}$$

It is not hard to see that

$$\begin{aligned}
\delta_{i-1} &= \frac{1}{2}(\delta_i + \delta_i^-) \\
\text{So } \delta_i^- &= 2\delta_{i-1} - \delta_i
\end{aligned}$$

Algorithm for S^A: Input $i, u = g(X)_{\{1,\ldots,i-1\}}$. The goal is to predict the i^{th} bit.
1. Choose $z \in_{\mathcal{U}} \{0,1\}^{l(n)-i+1}$, and form $y = uz$.
2. Evaluate $A(y)$.
3. If $A(y) = 1$, output z_1, the first bit of z.
Otherwise, output $1 - z_1$, the complement of the first bit of z.

Runtime of $S^A \approx T(n)$. The success probability of S^A is

$$
\begin{aligned}
& \Pr_{I,X}[S^A(I, g(X)_{\{1,\ldots,I-1\}} \text{ is correct }] \\
= \; & \frac{1}{l(n)} \sum_{i=0}^{l(n)-1} \Pr_X[S^A(I, g(X)_{\{1,\ldots,i-1\}} \text{ is correct }] \\
= \; & \frac{1}{l(n)} \sum_i [\frac{1}{2}\delta_i + \frac{1}{2}(1 - \delta_i^-)] \\
= \; & \frac{1}{l(n)} \sum_i [\frac{1}{2}\delta_i + \frac{1}{2}(1 - 2\delta_{i-1} + \delta_i)] \\
= \; & \frac{1}{l(n)} \sum_i [\delta_i - \delta_{i-1}] \\
= \; & \frac{1}{l(n)} [\delta_{l(n)} - \delta_0] \\
= \; & \frac{1}{l(n)} \delta(n)
\end{aligned}
$$

So the time success ratio of S^A is

$$l(n)\frac{T(n)}{\delta(n)} = l(n) \cdot (\text{time success ratio of } A)$$

$\square$

Math 267a - Foundations of Cryptography

Lecture #10: 3 February 1997

Lecturer: Sam Buss

Scribe Notes by: Imre Tuba

23 Stretching the output of a pseudorandom number generator

Goal: Given a PRN generator $g : \{0,1\}^n \to \{0,1\}^{n+1}$, construct a PRNG $h : \{0,1\}^n \to \{0,1\}^{p(n)}$ with $p(n) > n+1$.

g takes n really random bits and outputs $n+1$ pseudorandom bits.

Example 23.1

$$\begin{array}{rcl} x & = & \underbrace{101\ldots1}_{n \text{ bits}} \\ & & \quad\Downarrow g \\ g(x) & = & 0\underbrace{11\ldots1}_{n \text{ bits}} \\ & & \Downarrow \quad \Downarrow g \\ & & \underbrace{011\ldots1}_{n+2 \text{ bits}} \end{array}$$

Define:

$$\begin{array}{rcl} g^0(x) & = & x \\ g^1(x) & = & g(x) \\ g^2(x) & = & (g(x))_1 g(g(x)_{\{2,\ldots,n\}}) \\ & \vdots & \\ g^{i+1}(x) & = & (g(x))_1 g^i(g(x)_{\{2,\ldots,n\}}) \end{array}$$

Theorem 23.2 *Let $p(n)$ be a polynomial and $g : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRNG. Then $h(x) = g^{p(n)}(x)$ is a PRNG with $h : \{0,1\}^n \to \{0,1\}^{p(n)+n}$.*

Proof: h is polynomial time because g is. Let A be an adversary against h with runtime $T_A(n)$ and success probability $\delta_A(n)$. Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $Z \in_{\mathcal{U}} \{0,1\}^{n+p(n)}$. Let $Y_i \in_{\mathcal{D}_i} \{0,1\}^{n+p(n)}$ be a random variable with distribution $\mathcal{D}_i$ chosen as

$$Y_i = Z_{\{1,\ldots,p(n)-i\}} \underbrace{g^i(X)}_{n+i \text{ bits}} .$$

Let $\delta_i = \text{Prob}_{Y_i}[A(Y_i) = 1]$. The success probability of A is

$$\delta_A(n) = \underset{X}{\text{Prob}}[A(g^{p(n)}(X)) = 1] - \underset{Z}{\text{Prob}}[A(Z) = 1].$$

Note that

$$\underset{X}{\text{Prob}}[A(g^{p(n)}(X)) = 1] = \underset{Y_{p(n)}}{\text{Prob}}[A(Y_{p(n)}) = 1]$$

and that

$$\underset{Z}{\text{Prob}}[A(Z) = 1] = \underset{Y_0}{\text{Prob}}[A(Y_0) = 1].$$

Hence $\delta_A(n) = \delta_{p(n)} - \delta_0$.

We construct an adversary S^A for g. The algorithm for S^A is

S^A has input $u \in \{0,1\}^{n+1}$

1. Choose $i \in_{\mathcal{U}} \{1, \ldots, p(n)\}$.
2. Choose $z \in_{\mathcal{U}} \{0,1\}^{p(n)-I}$.
3. Form $\alpha = zu_1g^{i-1}(u_{\{2,\ldots,n+1\}})$.
4. Output $A(\alpha)$.

The runtime of S^A is $T_A(n) + n^{\mathcal{O}(1)} \approx T_A(n)$.

Let i be fixed for now. If $u \in_{\mathcal{U}} \{0,1\}^{n+1}$, then α is chosen with distribution $\mathcal{D}_{i-1}$. If $u = g(x)$ with $x \in_{\mathcal{U}} \{0,1\}^n$, then α has distribution $\mathcal{D}_i$.

The success probability of S^A is

$$\begin{aligned}
&\underset{X}{\text{Prob}}[S^A(g(X)) = 1] - \underset{U\in_{\mathcal{U}}\{0,1\}^{n+1}}{\text{Prob}}[S^A(U) = 1] \\
&= \frac{1}{p(n)} \sum_{i=1}^{p(n)} \left(\underset{X}{\text{Prob}}[S^A(g(X)) = 1 \mid i] - \underset{U\in_{\mathcal{U}}\{0,1\}^{n+1}}{\text{Prob}}[S^A(U) = 1 \mid i] \right) \\
&= \frac{1}{p(n)} \sum_{i=1}^{p(n)} \left(\underset{Y_i}{\text{Prob}}[A(Y_i) = 1] - \underset{Y_{i-1}}{\text{Prob}}[A(Y_{i-1}) = 1] \right) \\
&= \frac{1}{p(n)} \sum_{i=1}^{p(n)} (\delta_i - \delta_{i-1}) \\
&= \frac{1}{p(n)} (\delta_{p(n)} - \delta_0) \\
&= \frac{\delta_A(n)}{p(n)}
\end{aligned}$$

The time-success ratio of $S^A \approx \frac{p(n)T_a(n)}{\delta_A(n)} = p(n)\cdot$ (time-success ratio of A). □

We can construct an algorithm for encryption/decryption that uses a PRNG. The idea is to start with an n-bit long private key and use it as the seed of the PRNG, then stretch the output of the PRNG to $p(n)+n$ bits, where $p(n)$ is the length of the message that we want to encrypt.

If Alice wants to send a message to Bob they need to share a private key $x \in \{0,1\}^n$. Let $g : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ be a PRNG. Given the message $m = m_1 m_2 \ldots m_{p(n)}$, Alice's encryption algorithm is

Loop $i = 1, \ldots, p(n)$

$u = g(x)$
$e_i = u_1 \oplus m_i$
$x = u_{\{2,\ldots,n+1\}}$

End loop.

Output encrypted message: $e_1 e_2 \ldots e_{p(n)}$.

Bob's decryption algorithm is the same.

Math 267a - Foundations of Cryptography

Lecture #11: 5 February 1997

Lecturer: Sam Buss

Scribe Notes by: Bill Wood

24 Private Key Stream Cryptosystems

Private key stream cryptosystems encode messages one symbol at a time, as opposed to block systems which work with several symbols at once. As usual, we have Alice attempting to send a message to Bob with Eve trying to listen. The basic idea is that Ali ce has an encryption function $E : \{0, ..., n\} \times \{0,1\} \longrightarrow \{0,1\}$ such that $E(i, m_i)$ returns the ith bit e_i of the encryption of message m, so $e_i = E(i, m_i)$. (We may assume without loss of much generality that $m_i \in \{0,1\}$ and $e_i \in \{0,1\}$, i.e. that bits are coded to bits.) Similarly, Bob has a decryption function D such that $m_i = D(i, e_i)$. We present two versions of such a system.

Version One: Let $g : \{0,1\}^n \longrightarrow \{0,1\}^{p(n)}$ be a pseudorandom number generator and let Bob and Alice share a private key $x \in \{0,1\}^n$. Define

$$e_i = E_x(i, m_i) = m_i \oplus (g(x))_i,$$

where $(g(x))_i$ denotes the ith bi t of g(x). Since bitwise summing is a self-inversing operation, the decryption function D is identical to E :

$$m_i = D_x(i, e_i) = e_i \oplus (g(x))_i.$$

This system is good for sending up to $p(n)$ message bits.

Version Two: Let $g : \{0,1\}^n \longrightarrow \{0,1\}^{n+1}$ be a pseudorandom number generator. The encryption algorithm is as follows.

Loop for $i = 1, 2, 3, ...$
    Let $u = g(x)$
    Let $e_i = u_1 \oplus m_i$
    $x = u_{\{2,...,n+1\}}$
End Loop

As before, the decryption algorithm is identical. In fact, this is a merely a special case of the first version by the stretching theorem.

The stream cryptosystem has a few disadvantages. If a bit is skipped or dropped at any point in the transmission and the sender and receiver fall out of synch, the message will come back as gibberish. Also, Bob and Alice cannot share a key for two-way c ommunication using this system (although this problem can be rather easily solved if Alice and Bob each agree to use the even and odd numbered bits, respectively). The most significant problem, however, is that if Alice or Bob ever lose their value of x (say, by a computer crash), t hen the entire message must be resent and decoded from the beginning. That is, the ith bit cannot be decoded without doing the work for decoding the first $i-1$ bits. Later, we will discuss block cryptosystems which overcome these disadvantages.

25 Simple Passive Attack

Suppose adversary Eve knows Alice is sending one of two possible messages, either $0^{p(n)}$ or $1^{p(n)}$ (a string of all ones or all zeroes). Let the key $X \in_U \{0,1\}^n$ be chosen secretly by Alice and Bob, and suppose Alice secretly chooses a bit $B \in_U \{0,1\}$. Define the encryption as

$$e_X(B) = g(X) \oplus B^{p(n)}.$$

The adversary should try to predict B from $e_X(B)$. The *success rate* of the adversary is defined as

$$\begin{aligned}\delta(n) &= \Pr_{X,B}[A(e_X(B)) = B] - \frac{1}{2} \\ &= \frac{1}{2}(\Pr_{X,B}[A(e_X(B)) = B] - \Pr_{X,B}[A(e_X(B)) \neq B]).\end{aligned}$$

This is simply one-half the probability that the adversary is right minus one-half the probability the adversary is wrong. The cryptosystem defined is $R(n)$-*secure* provided every adversary A has time-success ratio greater than or equal to $R(n)$. It is *secure* provided it is n^c-secure for all $c > 0$.

Theorem 25.1 *If g is a pseudorandom number generator, then the cryptosystem described above is secure.*

Proof: Let A be an adversary against the cryptosystem with $g : \{0,1\}^n \longrightarrow \{0,1\}^{p(n)}$ a pseudorandom number generator. We construct an adversary S^A against g. The input to S^A is $z \in \{0,1\}^{p(n)}$, and the algorithm is as follows:

Algorithm for S^A. The input is $z \in \{0,1\}^{p(n)}$
Choose $b \in_U \{0,1\}$
Let $e = b^{p(n)} \oplus z$
If $A(e) = b$, then output 1. Otherwise, output 0.

The runtime of S^A is approximately that of A. For the success probablity, we examine the two cases.
Case One: If $Z \in_U \{0,1\}^{p(n)}$, then $\Pr_Z[S^A(Z) = 1] = \frac{1}{2}$ because e is independent of b and $b = 1$ with probability $\frac{1}{2}$.

Case Two: If $Z = g(X), X \in_U \{0,1\}^n$, then

$$\Pr_X[S^A(g(X)) = 1] = \Pr_{X,B}[A(e_X(B)) = B] = \delta_A(n) + \frac{1}{2}.$$

Thus, the success probability of S^A is $\delta_A(n) + \frac{1}{2} - \frac{1}{2} = \delta_A(n)$, the success probability of A. The time-success ratio of S^A is therefore equal to that of A, as desired. □

Homework 9 Prove this theorem under the assumption that an arbitrary pair of $p(n)$-bit messages (which may be chosen by Eve) are to be distinguished (as opposed to just $0^{p(n)}$ and $1^{p(n)}$).

26 Simple Chosen Plaintext Attacks (Informal Definition)

In this situation, the adversary forces Alice to send some messages chosen adaptively by the adversary. The adversary then chooses two new messages m_0 and m_1, of which Alice secretly chooses one at random and encrypts it as e. From e, the adve rsary tries to determine which of m_0 and m_1 was sent.

The simple chosen plaintext attack is defined formally in the next lecture.

Math 267a - Foundations of Cryptography

Lecture #12: 7 February 1997

Lecturer: Sam Buss

Scribe Notes by: David Meyer

27 Simple chosen plaintext attack

Last time we saw the informal definition of a simple chosen plaintext attack; today we formalize the definition.

Definition 27.1 *Alice and Bob (who is irrelevant to this discussion) share a (publicly known) pseudorandom number generator* $g : \{0,1\}^n \longrightarrow \{0,1\}^{k(n)+p(n)}$ *and a private key* $x \in \{0,1\}^n$*. ($k(n)$ will be the number of plaintext bits Eve will force Alice to encode; $p(n)$ will be the length of the final message Eve tries to decode.) Their adversary Eve has three algorithms:*

(i) $M : \{1,\ldots,k(n)\} \times \{0,1\}^{k(n)-1} \longrightarrow \{0,1\}$

(ii) $P : \{0,1\}^{k(n)} \longrightarrow \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$

(iii) $A : \{0,1\}^{k(n)} \times \{0,1\}^{p(n)} \longrightarrow \{0,1\}$

which she uses to make a simple chosen plaintext attack*:*

> **For** $i = 1,\ldots,k(n)$,
>
> > *Eve computes* $m_i = M(i, e_1 \cdots e_{i-1} 0^{k(n)-i})$.
> > *Alice computes* $e_i = m_i \oplus (g(x))_i \quad \big(= E_x(i, m_i)\big)$.
> > *Alice reveals* e_i *to Eve.*
>
> **End for**.
>
> *Eve computes* $(m_0^*, m_1^*) = P(e_1 \cdots e_{k(n)})$.
>
> *Alice privately chooses* $j \in \{0,1\}$ *and sends* m_j^* *encrypted as*
>
> $$e^* = m_j^* \oplus (g(x))_{\{k(n)+1,\ldots,p(n)+k(n)\}}.$$
>
> *Finally Eve runs* $A(e^*)$ *to try to determine the value of* j.

Eve's success rate is $\Pr_X[A(e^*) = j] - 1/2$ and her runtime is the total runtime of the $k(n)$ invocations of M plus the runtimes of P and A.

Definition 27.2 *The cryptosystem is $S(n)$*-secure against simple chosen plaintext attack *if every (M, P, A) adversary has time/success ratio at least $S(n)$. It is* secure against simple chosen plaintext attack *iff it is n^c-secure for all $c > 0$.*

Theorem 27.3 *If g is a pseudorandom number generator then the above (stream private key) cryptosystem is secure against simple chosen plaintext attack.*

Exercise 10 Prove the theorem. Hints: Simplify the problem by letting Alice just give the first $k(n)$ bits to Eve rather than going through the charade of the for-loop in the attack. Then proceed as in the proof from last time.

Luby gives some subtle variations on chosen plaintext attack in Chapter 11, none of which changes the basic intuition. We can even think of Eve as being temporarily able to see both the plain and the encoded messages; then the theorem is that once she can no longer see the plain messages, having seen them gives her no information useful for decrypting subsequent encrypted messages.

28 Block cryptosystems

The 'block' in 'block cryptosystems' refers to the fact that a whole block of bits is encoded at once. Blocks will have 'indices' (possibly just timestamps) and we will assume that the i^{th} block can be encrypted/decrypted without having to handle the 'previous' $i - 1$ blocks (which is useful for synchronization). That is, the runtime to encrypt/decrypt the i^{th} block by itself should include a factor of $|i|^{O(1)}$, not $i^{O(1)}$. Note that this latter condition (of independence from previous blocks) is often not satisfied by systems called block cryptosystems elsewhere.

To implement a block cryptosystem, we will need to define a pseudorandom *function* generator. We begin with some notation:

Definition 28.1 *Let n be a security parameter and $\ell(n), k(n) \geq n$ be polynomials. A* random function $f : \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$ *is chosen by setting all the $2^{\ell(n)}$ many values of f to randomly chosen values in $\{0,1\}^{k(n)}$. There are $(2^{k(n)})^{2^{\ell(n)}} = 2^{k(n)\cdot 2^{\ell(n)}}$ many such functions. We will denote the space of all such functions by* $\mathbf{Fnc}\colon \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$ *and consider random functions $F \in_{\mathcal{U}} \mathbf{Fnc}\colon \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$.*

Our goal is to be able to pick pseudorandom functions from this space which work as well as truly random F.

Remark 28.2 Notice that any random function can be described by a single (exponentially long) bit string, that is, by a random number. A pseudorandom number generator, used with this interpretation, would *not* suffice for our purposes, however, because it could not generate exponentially long bit strings.

Now let $f : \{0,1\}^n \times \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$ be polynomial time. The first input to f is a private key $x \in \{0,1\}^n$. Let $f_x(u) = f(x,u)$ for $u \in \{0,1\}^{\ell(n)}$. Then $f_x \in \mathbf{Fnc}\colon \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$.

Definition 28.3 *An* adversary against f_x *is an* oracle *Turing machine* A^g*, meaning* A *running with* $g : \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$ *available as a 'black box'—*A *can (repeatedly) compute* $u \in \{0,1\}^{\ell(n)}$ *and obtain* $g(u)$ *in return. The* success rate *of* A *is*

$$\delta(n) = \left|\Pr_X[A^{f_X}(n) = 1] - \Pr_F[A^F(n) = 1]\right|,$$

where $X \in_{\mathcal{U}} \{0,1\}^n$ *and* $F \in_{\mathcal{U}}$ $\mathbf{Fnc}\colon \{0,1\}^{\ell(n)} \longrightarrow \{0,1\}^{k(n)}$.

Math 267a - Foundations of Cryptography

Lecture #13: 10 February 1997

Lecturer: Sam Buss

Scribe Notes by: Roland Meyer

29 Pseudo random function generators

Construction of a block cryptosystem using a pseudo random function generator

Let $f : \{0,1\}^n \times \{0,1\}^{l(n)} \to \{0,1\}^{k(n)}$. For our block cryptosystems

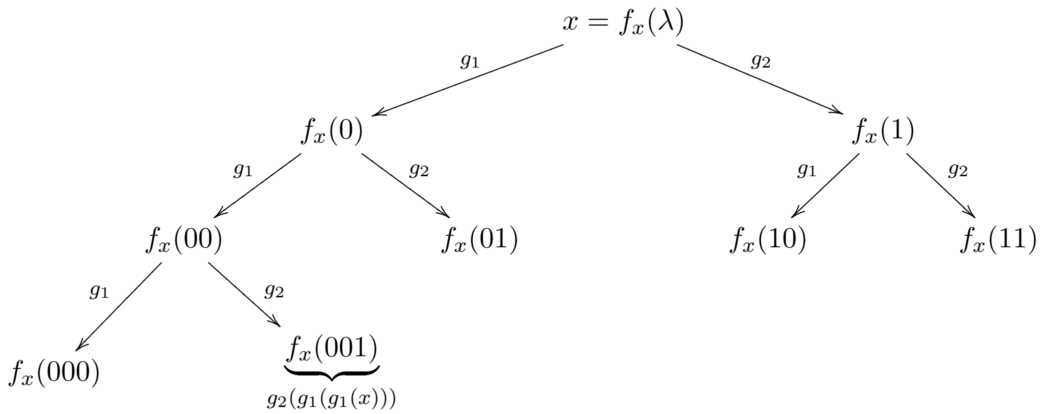
- plain text messages are $m \in \{0,1\}^{k(n)}$,
- indices are $i \in \{0,1\}^{l(n)}$,
- the secret key is $x \in \{0,1\}^n$,
- the encrypted message is $e = f(x,i) \oplus m = E_x(i,m)$,
- the decryption algorithm is the same as the encryption algorithm and
- $m = f(x,i) \oplus e = D_x(i,e)$.

Indices should never be reused! The transmitted text is the pair $\langle e, i\rangle$.

Homework 11: Formulate good notions of simple passive/simple plain text attacks. Prove that if f is a pseudo random function generator, the above cryptosystem is secure against these attacks.

Theorem 29.1 *If there is a pseudo random number generator g then there is a pseudo random function generator f.*

Proof: By the earlier Stretching Theorem, we may assume without loss of generality that $g : \{0,1\}^n \to \{0,1\}^{2n}$. We define $f(x,u) = f_x(u)$. In fact we will define $f(x,u)$ for all $x \in \{0,1\}^n$ and all $u \in \{0,1\}^{\leq l(n)}$. Let λ be the empty string in $\{0,1\}^*$ and $f_x(\lambda) = x$. Let $g(u) = g_1(u)g_2(u)$, where $g_1(u), g_2(u) \in \{0,1\}^n$. Let $f_x(u0) = g_1(f_x(u))$ and $f_x(u1) = g_2(f_x(u))$.



Let A be an adversary against f, $T_A(n)$ be the runtime of A and $\delta_A(n)$ be the success probability of A. We will construct an adversary S^A against g with time success ratio

$$R_{S^A}(n) \leq n^{O(1)}(R_A(n))^2, (\text{where } R_A(n) = T_A(n)/\delta_A(n)).$$

Let $\delta_0 = \Pr_X[A^{f_x}(n) = 1]$ and $\delta_1 = \Pr_F[A^F(n) = 1]$ where $X \in_{\mathcal{U}} \{0,1\}^n$ and $F \in_{\mathcal{U}} \mathrm{Fnc} : \{0,1\}^{l(n)} \to \{0,1\}^n$. Then $\delta_A(n) = \delta_0 - \delta$. The adversary S^A will simulate A's computation. S^A has an element $y \in \{0,1\}^{2n}$ as input and A has a function of type $\{0,1\}^{l(n)} \to \{0,1\}^n$ as input which it is allowed to use as a "black-box" computational device.

We would like to simulate A on input f_x where $x = g^{-1}(\text{input to } S^A)$ and/or on input F chosen at random.

We will arrange for a smooth transition between simulating A^{f_x} and A^F. During the course of computation, A^F makes up to $T_A(n)$ many queries to F. Let A^F query values $F(w_1), F(w_2), \ldots, F(w_{T_A(n)})$. In the simulation, S^A will set values for $F(v_1), F(v_2), \ldots, F(v_{(3/2)T_A(n)l(n)})$ so that whenever a value for $F(u0)$ or $F(u1)$ is set by A, a value for $F(u)$ was previously set and both $F(u0)$ and $F(u1)$ are computed. A remembers and does not recompute these values. Let $m(n) = T_A(n)l(n)$ be an upper bound on the number of times S^A sets values for a pair $F(u0), F(u1)$. S^A will pick a parameter $k \in \mathbb{N}$.

Let $u_1, \ldots u_{m(n)}$ be the values such that S^A sets $F(u_i0), F(u_i1)$. S^A sets $F(0)$ at random in $\{0,1\}^n$. If $i < k$, S^A chooses these two values at random in $\{0,1\}^n$. If $i = k$ S^A chooses $F(u_i0) = y_{\{1,\ldots,n\}}$ and $F(u_i0) = y_{\{n+1,\ldots,2n\}}$. If $i > k$, S_A chooses $F(u_i0) = g_1(F(u_i))$ and $F(u_i1) = g_2(F(u_i))$.

<u>Algorithm for S^A:</u>

1. choose $k \in \{1, \ldots, m(n)\}$
2. simulate A as above
3. Output whatever the simulation of A outputs.

The runtime of S^A is approximately $T_A(n)$. Let

$$P_{0,k} = \Pr_X[S^A(g(X)) = 1 \mid k]$$

$$P_{1,k} = \Pr_{Y \in \{0,1\}^{2n}}[S^A(Y)) = 1 \mid k]$$

By inspection, one can see that $\delta_0 = P_{0,1}$, $\delta_1 = P_{1,m(n)}$ and $P_{0,k+1} = P_{1,k}$. Furthermore

$$\Pr_X[S^A(g(X)) = 1] = \frac{1}{m(n)} \sum_{i=1}^{m(n)} P_{0,i}$$

and

$$\Pr_Y[S^A(Y) = 1] = \frac{1}{m(n)} \sum_{i=1}^{m(n)} P_{1,i}$$

Thus the success probability of S^A is

$$\frac{1}{m(n)} \sum_{i=1}^{m(n)} (P_{0,i} - P_{1,i}) = \frac{1}{m(n)}(P_{0,1} - P_{1,m(n)}) = \frac{1}{m(n)}(\delta_0 - \delta_1) = \frac{1}{m(n)}\delta_A(n)$$

and the success ratio of S^A is approximately

$$\frac{T_A(n)}{\frac{1}{m(n)}\delta_A(n)} = \frac{T_A(n)l(n)T_A(n)}{\delta_A(n)} \leq n^{O(1)}(R_A(n))^2$$

$\square$

Math 267a - Foundations of Cryptography

Lecture #14: 12 February 1997

Lecturer: Sam Buss

Scribe Notes by: Christian Gromoll

30 Trapdoor functions & RSA

Probability distributions for trapdoor and public keys are given by a P-time function

$$D_n : \{0,1\}^n \longrightarrow \{0,1\}^{m(n)} \times \{0,1\}^{l(n)}.$$

If $D_n(y) = \langle x, z\rangle$, then x is the private *trapdoor* key, and z is the *public* key. This gives a distribution $\mathcal{D}_n$ on $\{0,1\}^{m(n)} \times \{0,1\}^{l(n)}$, namely $u \in_{\mathcal{D}_n} \{0,1\}^{m(n)} \times \{0,1\}^{l(n)}$ is given by

$$u = D_n(v),$$

where $v \in_{\mathcal{U}} \{0,1\}^n$. Then a *trapdoor function* is an

$$f : \{0,1\}^{l(n)} \times \{0,1\}^n \longrightarrow \{0,1\}^{k(n)},$$

which satisfies the two conditions below. Note that here the first input is the public key, and the second input is the "plain-text" message.

Let $\langle x, z\rangle$ be trapdoor and public keys.

Condition 30.1 There is a P-time function $g : \{0,1\}^{l(n)} \times \{0,1\}^{k(n)} \times \{0,1\}^{m(n)} \longrightarrow \{0,1\}^n$ such that

$$f(z, g(z, f(z, y), x)) = f(z, y)$$

or, writing $f_z(y) = f(z, y)$,

$$f_z(g(z, f_z(y), x)) = f_z(y).$$

Condition 30.2 (One-way condition) It is difficult to invert f_z without knowing x.

The second condition is formalized in the following

Definition 30.3 *An* adversary *for a trapdoor function f is an $A : \{0,1\}^{l(n)} \times \{0,1\}^{k(n)} \longrightarrow \{0,1\}^n$ with success probability*

$$\Pr_{X,Y,Z}[f_Z(A(Z, f_Z(Y))) = f_Z(Y)].$$

We say $\mathcal{D}_n, f$ form an $S(n)$-secure trapdoor system *if and only if every adversary A has time-success ratio $\geq S(n)$; f is called a* trapdoor function *if and only if $\mathcal{D}_n, f$ form an n^c-secure trapdoor system, for all $c > 0$,*

Example 30.4 RSA (root extraction). Let

$$D_n(y) = \left\langle \underbrace{\langle p, q \rangle}_{\substack{\text{private}\\ \text{key}}} \; , \; \underbrace{\langle pq, e \rangle}_{\substack{\text{public}\\ \text{key}}} \right\rangle$$

where p, q, e are n-bit primes chosen at random, with e relatively prime to $(p-1)(q-1)$, i.e. $e \in_{\mathcal{U}} \mathbb{Z}^*_{pq}$ (which is of order $(p-1)(q-1)$). Define

$$f_{\langle pq,e \rangle}(m) = m^e \bmod pq.$$

Here m is relatively prime to pq with extremely high probability, so we just assume it is. Now given knowledge of p and q, one can invert $f = f_{\langle pq,e \rangle}$ as follows:

Use Euclid's algorithm to find a $d > 0$, $d \in \mathbb{N}$ such that

$$de \equiv 1 \bmod (p-1)(q-1).$$

Remark 30.5 Euclid's algorithm gives a c, d such that

$$de + c(p-1)(q-1) = 1.$$

If $d < 0$, we have also

$$((p-1)(q-1) - d)e + (c-e)(p-1)(q-1) = 1.$$

Euclid's algorithm can easily be computed in polynomial time.

Once d is known, let

$$g(\langle pq, e \rangle, v, \langle p, q \rangle) = v^d \bmod pq.$$

We have to check that $(f(m))^d \equiv m \bmod pq$. Working $\bmod pq$,

$$\begin{aligned} f(m) &= m^e, \quad \text{so} \\ (m^e)^d &= m^{ed} = m^{e(p-1)(q-1)+1} \\ &= m \end{aligned}$$

since $m^{(p-1)(q-1)} = 1 \pmod{pq}$. Notice that the private key could be taken to be just d, and p, q don't need to be remembered.

Usages:

1. Bob sends a message to Alice. Bob knows the public information, i.e. pq, e. To send a message $m \in \mathbb{Z}_{pq}^*$, Bob encrypts it as
$$m' = f(m) = m^e \bmod pq.$$
Bob sends m' to Alice over a public line. Alice decrypts it as
$$m = (m')^d \bmod pq.$$
This would be used in a block cryptosystem, with m being one block of the message.

2. Alice can send authenticated messages to Bob, i.e. Bob can be sure they come from Alice. Alice wants to send a message $m \in \mathbb{Z}_{pq}^*$. She encrypts it as
$$m' = m^d \bmod pq.$$
Bob then decrypts it as
$$m = (m')^e \bmod pq$$
using the publicly known pq, e.

3. The above two can be combined as follows: Alice and Bob both choose public and private keys $\langle p_A, q_A \rangle, e_A; d_A$ and $\langle p_B, q_B \rangle, e_B; d_B$ respectively. Alice sends a message m to Bob with
$$m' = \left(m^{d_A} \bmod p_A q_A\right)^{e_B} \bmod p_B q_B.$$
Bob decrypts this as
$$m = \left((m')^{d_B} \bmod p_B q_B\right)^{e_A} \bmod p_A q_A.$$
In this scheme, Bob can be sure that the message came from Alice, while Alice can be sure that only Bob can read the message.

Math 267a - Foundations of Cryptography

Lecture #15: 14 February 1997

Lecturer: Sam Buss

Scribe Notes by: Tin Yen Lee

31 Square Root Extraction

Square Root Extraction is used as a trapdoor function.

Set-up:
Choose 2 n-bit primes p and q.
$D_n(y) = \langle\langle p, q\rangle; pq\rangle$
The encryption function is $f_{pq}(x) = x^2 \bmod pq$

To invert this, we have a homework problem which says: "Given p, q, and $z = x^2$, we can find the 4 values a_1, a_2, a_3, a_4 such that $a_i^2 \equiv z \bmod pq$."
Thus, if we have knowledge of the private key $\langle p, q\rangle$, we can invert f_{pq}.

Problem: We have 4 square roots, instead of just 1.
Solution #1: Assume that only 1 of the roots will be a valid "message".
Solution #2: Take $p, q \equiv 3 \bmod 4$. Then there is a unique square root which is a quadratic residue mod pq.
Note: RSA is more flexible and more widely used. The advantage of square root extraction is that we know that the statement "If finding square roots is easy, then factoring is easy" is provable.

32 Existence of Pseudorandom Number Generators

In the next few weeks, we will be refering to Luby, Lectures 5-10.
Goal: To prove that if one-way functions exist, then pseudorandom number generators exist.

Definition 32.1 *Let $X_1, \ldots X_m$ be random variables, taking values in a finite set S. $X_1, \ldots X_m$ are* pairwise independent **iff**

$$\forall \alpha, \beta \in S,\ \Pr[X_i = \alpha, X_j = \beta] = \Pr[X_i = \alpha] \Pr[X_j = \beta]\ \forall i, j,\ i \neq j, i, j \in \{1, \ldots, m\}$$

or, equivalently,

$$\forall\, \alpha, \beta, i \neq j,\ \Pr[X_i = \alpha | X_j = \beta] = Pr[X_i = \alpha]$$

.

Example #1: Generalized Inner Product Space
We'll choose values in $S = \{0,1\}^n$. $X_1, \ldots, X_m$ will be pairwise independent.
Let $l = \lceil \log_2(m+1) \rceil$ = number of bits in the binary representation of m.
Choose $X_1, \ldots, X_m$ as follows:
Choose $a_1, \ldots, a_l \in_{\mathcal{U}} \{0,1\}^n$.
Set $X_i = \bigoplus_{k:i_k=1} a_k$, where i_k = value of the kth bit of i's binary representation.

Claim: These are pairwise independent random variables.

Proof of Claim:
Fix $i \neq j$.
Let $I_0 = \{k | i_k = 1 \text{ and } j_k = 0\}$
Let $I_1 = \{k | i_k = 1 \text{ and } j_k = 1\}$
Let $I_2 = \{k | i_k = 0 \text{ and } j_k = 1\}$

Then

$$X_i = \bigoplus_{k \in I_0} a_k \ \oplus\ \bigoplus_{k \in I_1} a_k$$

$$X_j = \bigoplus_{k \in I_2} a_k \ \oplus\ \bigoplus_{k \in I_1} a_k$$

Without loss of generality, we may assume that $I_2 \neq \emptyset$, since $i \neq j$. Now

$$X_j = \bigoplus_{k \in I_2} a_k \ \oplus\ (X_i \ \oplus\ \bigoplus_{k \in I_0} a_k)$$

It is easy to see that $\bigoplus_{k \in I_2} a_k$ is independent of X_i and is independent of $(X_i \ \oplus\ \bigoplus_{k \in I_0} a_k)$; thus, it is also uniformly distributed in $\{0,1\}^n$.
Therefore, X_j is independent of X_i and $X_j \in_{\mathcal{U}} \{0,1\}^n$.
QED Proof of Claim

Note: We can think of $i = (i_l, i_{l-1}, i_{l-2}, \ldots, i_1)$ as a row l-vector
Let A be an l by n matrix with rows a_i. Then $X_i = iA$.
When n=1, this is a dot product, so in this sense we have a Generalized Inner Product Space.

Remarks:

1) $X_1, \ldots, X_m$ are not independent since, for example, $X_3 = X_1 \oplus X_2$ (=$a_1 \oplus a_2$)

2) We needed only $n \log(m)$ many random bits to choose $a_1, \ldots, a_l$. On the other hand, to choose $X_1, \ldots, X_m$ independently, we would need $n \cdot m$ many random bits.

Example #2 Linear Polynomial Space
Fix a finite field F. Without loss of generality, the elements of F are $S = \{0, 1, \ldots, f-1\}$. To choose $X_0, \ldots, X_{f-1}$, pairwise independent, choose $a, b \in_{\mathcal{U}} F$. Let $X_i = a * i + b$, with * and + being the field operations.

Claim: $X_0, \ldots, X_{f-1}$ are pairwise independent random variables.
Proof of Claim:
Fix $i \neq j$. Fix α, β.
$X_i = \alpha$ and $Y_j = \beta$ hold iff $\alpha = a * i + b$ and $\beta = a * j + b$,
and for every pair α, β there are unique values for a and b such that these hold.
$\Pr[X_i = \alpha \text{ and } X_j = \beta] = \frac{1}{f^2} = \frac{1}{f} \cdot \frac{1}{f} = \Pr[X_i = \alpha] Pr[X_j = \beta]$.
QED Proof of Claim

Note: The number of random bits used to choose a and $b = 2\log_2(f)$
Homework 12: Generalize the above example to get k-wise independent $X_0, \ldots, X_{f-1}$.
Hint: degree k polynomials.

Example #3: Modulo Prime Space
This is a special case of Example #2 with $F = GF_p$, p a prime.
$X_i = a * i + b (mod p)$.

Math 267a - Foundations of Cryptography

Lecture #16: 19 February 1997

Lecturer: Sam Buss

Scribe Notes by: Anand Desai

33 Some Probability Review

33.1 Some Simple Definitions

Example 33.1 Let X be a {0,1}-value random variable, with $Pr[X = 1] = p$.
The *expected value* of X is

$$\begin{aligned} E[X] &= \sum_{i \in \{0,1\}} i \cdot Pr[X = i] \\ &= 1 \cdot p + 0 \cdot (1 - p) = p \end{aligned}$$

The *variance* of X is

$$\begin{aligned} Var(X) &= E[(X - E[X])^2] \\ &= p(1 - p)^2 + (1 - p)(0 - p)^2 = p(1 - p) \end{aligned}$$

The *standard deviation* of X is

$$\sigma = \sqrt{Var(X)}$$

Homework 13. Let X be a $\{-1, 1\}$-valued random variable. Let $p = Pr[X = 1]$. Show the following:

1. $E[X] = 2p - 1$
2. $Var(X) = 4p(p - 1)$
3. $Var(X) \leq 1$

33.2 Markov Inequality

If X is a nonnegative random variable and $\delta > 0$, then

$$Pr[X \geq \delta] \quad \leq \quad \frac{E[X]}{\delta}$$

Proof:

$$\begin{aligned} E[X] &\geq \delta \cdot Pr[X \geq \delta] + 0 \cdot Pr[X < \delta] \\ &\geq \delta \cdot Pr[X \geq \delta] \end{aligned}$$

$\square$

Homework 14. State and prove some general forms of Markov's Inequality where $X \geq a$ (and $X \leq a$) instead of $X \geq 0$.

33.3 Chebychev Inequality

Let X be a real-value random variable. Let $\delta \geq 0$. Then,

$$Pr\left[|X - E[X]| \geq \delta\right] \quad \leq \quad \frac{Var(X)}{\delta^2} = \frac{\sigma^2}{\delta^2}$$

Proof:

$$\begin{aligned} Pr\left[|X - E[X]| \geq \delta\right] &= Pr\left[(X - E[X])^2 \geq \delta^2\right] \\ &\leq \frac{E[(X - E[X])^2]}{\delta^2} \\ &\leq \frac{Var(X)}{\delta^2} \end{aligned}$$

$\square$

33.4 Chernoff Bounds

Let $X_1, \ldots, X_m$ be independent $0-1$ valued random variables with, $Pr[X_i = 1] = p$.
Let $0 < \delta < p(1-p)$. Then,

$$\begin{aligned} Pr\left[\left|\frac{1}{n}\sum X_i - p\right| \geq \delta\right] &\leq 2e^{-\frac{\delta^2 n}{2\sigma^2}} \\ &\leq \frac{2}{\left(e^{\frac{\delta^2}{2\sigma^2}}\right)^n} \end{aligned}$$

33.5 Pairwise Independent Sampling Theorem

Let $X_1, \ldots, X_m$ be pairwise independent random variables with identical expected value μ and variance σ^2. Let $0 < \delta$. Then,

$$Pr\left[\left|\left(\frac{1}{n}\sum X_i\right) - \mu\right| \geq \delta\right] \leq \frac{\sigma^2}{\delta^2 n} = \left(\frac{\sigma^2}{\delta^2}\right) \cdot \frac{1}{n}$$

Proof: Wlog we can take $\mu = 0$, since otherwise we could replace X_i with $X_i - \mu$. Apply Chebychev Inequality to $Y = \frac{1}{n}\sum X_i$

$$\begin{aligned} Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n} X_i\right| \geq \delta\right] &\leq \frac{Var(\frac{1}{n}\sum X_i)}{\delta^2} \\ &\leq \frac{E[(\frac{1}{n}\sum X_i)^2]}{\delta^2} \\ &\leq \frac{E[(\sum X_i)^2]}{n^2\delta^2} \end{aligned}$$

$$\begin{aligned} E[(\sum X_i)^2] &= E[\sum_i X_i{}^2 + \sum_{i \neq j} X_i X_j] \\ &= E[\sum_i X_i{}^2] + \sum_{i \neq j} E[X_i X_j] \\ &= E[\sum_i X_i{}^2] + \sum_{i \neq j} E[X_i] \cdot E[X_j] \\ &= \sum_i E[X_i{}^2] + \sum_{i \neq j} 0 \cdot 0 \\ &= n \cdot \sigma^2 \end{aligned}$$

Now we have,

$$Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n} X_i\right| \geq \delta\right] \leq \frac{n\sigma^2}{n^2\delta^2} = \frac{\sigma^2}{\delta^2 n}$$

□

Homework 15. Recall from Day 2 our proof that $RP(\frac{1}{n}) = RP(\frac{1}{2})$. Redo that proof with pairwise independent sampling. Compare the amount of randomness and the number of samples used now to that of the Day 2 proof.

34 Hidden Inner Product Bit

Definition 34.1 If $u, v \in \{0,1\}^n, u = u_1 \ldots u_n, v = v_1 \ldots v_n$, then

$$u \bullet v = \sum u_i v_i \bmod 2$$

Definition 34.2 [Hidden Inner Product Bit] Let $f : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$ be a polynomial time function ensemble. Let $x \in \{0,1\}^n$ be the private input to f. Let $z \in \{0,1\}^n$ be a public value. The inner product bit of f w.r.t. z is $x \bullet z$.

Let $A = \{0,1\}^{l(n)} \times \{0,1\}^n \longrightarrow \{0,1\}$ be an adversary. The success rate of A is

$$\begin{aligned}\delta(n) &= \Pr_{X,Z}[A(f(X),Z) = X \bullet Z] - \Pr_{X,Z}[A(f(X),Z) \neq X \bullet Z] \\ &= 2\left(\Pr_{X,Z}[A(f(X),Z) = X \bullet Z] - \frac{1}{2}\right)\end{aligned}$$

The time-success ratio of A is defined as usual.
The inner product bit is $R(n) - secure$ iff every adversary has time-success ratio $\geq R(n)$
The inner product bit is *hidden* if it is $n^c - secure$ for all $c > 0$.

Theorem 34.3 (Hidden-Bit Theorem) *Suppose* $f : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$ *is a one-way function, then the inner product bit of* f *is hidden.*

Proof: Given in the next lecture. □

Homework 16. Give an example of an f for which the inner product bit is hidden but f is not one-way. [Hint: Think "trivial"]

Corollary 34.4 *Let* $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ *be a one-way permutation. Then* $g : (x,z) \mapsto (f(x), z, x \bullet z)$ *is a pseudorandom number generator.* $g : \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n+1}$

Proof: It suffices to show g is *next-bit-unpredictable*. For $i \leq n$, it is clearly impossible to predict the ith bit from the previous bits, with a probability $> \frac{1}{2}$ (since if $x \in_{\mathcal{U}} \{0,1\}$ and given f is a permutation, then $f(x) \in_{\mathcal{U}} \{0,1\}^n$.)

Finally: the $(2n+1)$th bit is unpredictable since it is the hidden inner product bit. □

Math 267a - Foundations of Cryptography

Lecture #17: 21 February 1997

Lecturer: Sam Buss

Scribe Notes by: David Little

35 More on Hidden Bits

Reminder of the definition of success probability $\delta(n)$ against "hidden bit"

$$\delta(n) = \Pr_{X,Z}[A(f(X), Z) = X \bullet Z] - \Pr_{X,Z}[A(f(X), Z) \neq X \bullet Z]$$

Notation $\{-1, 1\}$ representation of Boolean values

Let $x \in \{0, 1\}$. Define $\overline{x} = (-1)^x = 1 - 2x$ (i.e. $\overline{0} = 1, \overline{1} = -1$).

Using this notation, we have $\overline{x \oplus y} = \overline{x} \cdot \overline{y}$

Theorem 35.1 *(Hidden Bit Theorem)*

If $f(x)$ is a one-way function then the inner product bit of f is hidden.

Lemma 35.2 *(Hidden Bit Technical Lemma)*

Let $B : \{0, 1\}^n \to \{0, 1\}$ be a function ensemble with runtime $T_B(n)$.

Let $Z \in_{\mathcal{U}} \{0, 1\}^n$ and let $x \in \{0, 1\}^n$

Define $\delta_x^B := \Pr_Z[B(Z) = x \bullet Z] - \Pr_Z[B(Z) \neq x \bullet Z] = E_Z[\overline{B(Z)} \cdot \overline{x \bullet Z}]$

Then there is an oracle adversary S^B which on input $\delta > 0$ outputs a list L s.t. $\forall x \in \{0, 1\}^n$ $\delta_x^B \geq \delta \Rightarrow \Pr[x \in L] \geq \frac{1}{2}$ and the running time of S^B is $n^{O(1)} T_B(n)/\delta^2$.

Proof: (of Technical Lemma)

Let $A(-, -)$ be an adversary against the inner product bit of f. A has runtime $T(n)$ and success probability $\delta(n)$ We want to construct an adversary of f.

Define $\delta_x^A := E_Z[\overline{A(f(x), Z)} \cdot \overline{x \bullet Z}]$ (which gives the following identity $E_X[\delta_x^A] = \delta(n)$)

By Markov's inequality and the fact that $\delta_x^A \leq 1$, we have

$$\Pr_X[\delta_x^A \geq \frac{\delta(n)}{2}] \geq \frac{\delta(n)}{2}$$

Let B_y be the mapping $z \mapsto A(y, z)$. The following is an algorithm for an oracle A_f against f. It accepts as input $y = f(x)$ and attempts to find $f^{-1}(y)$.

1. Run $S^{B_y}\left(\frac{\delta(n)}{2}\right)$ from the technical lemma
2. Check to see if there exists an x' on the list output by step 1 that satisfies $f(x') = y$. If so, then output x'. Otherwise algorithm fails.

Runtime of A_f is approximately the same as runtime of $S^{B_y} = n^{O(1)}T(n)/\delta(n)^2$

Success probability of A_f (over random values of x) $\geq \frac{\delta(n)}{2} \cdot \frac{1}{2} = \frac{\delta(n)}{4}$

Time-success ratio of A_f $n^{O(1)}T(n)/\delta(n)^3$ □

Describe S^B's algorithm

Assume $\delta_x^B > \delta$ and let $x \in \{0,1\}^n$
Let $e_i = (\overbrace{0,\ldots,0}^{i-1}, 1, \overbrace{0,\ldots,0}^{n-i}) \in \{0,1\}^n$
Let $\mu_i = \delta_x^B \cdot \overline{x_i} = \begin{cases} \delta_x^B \text{ if } x_i = 0 \\ -\delta_x^B \text{ if } x_i = 1 \end{cases}$ where x_i is the i^{th} bit of x.
In order to find x, we need to find all of the x_i's, which is equivalent to finding the signs of the μ_i's. We begin by fixing i.

$$\delta_x^B = E_Z[\overline{B(Z)} \cdot \overline{(x \bullet Z)}] = E_Z[\overline{B(e_i \oplus Z)} \cdot \overline{(x \bullet (e_i \oplus Z)})]$$

$$\overline{x \bullet (e_i \oplus Z)} = \overline{(x \bullet e_i) \oplus (x \bullet Z)} = \overline{(x \bullet e_i)} \cdot \overline{(x \bullet Z)} = \overline{x_i} \cdot \overline{(x \bullet Z)}$$

Therefore $\delta_x^B = \overline{x_i} \cdot E_Z[\overline{B(e_i \oplus Z)(x \bullet Z)}]$ and $\mu_i = E_Z[\overline{B(e_i \oplus Z)(x \bullet Z)}]$

We'll estimate the values μ_i by sampling values $T_1, \ldots, T_m$ for Z and averaging. $T_1, \ldots, T_m$ are choosen pairwise independent with the generalized Inner Product Construction.

Let $m = \lceil 2n/\delta^2 \rceil$ and $l = \lceil log_2(m+1) \rceil$
Choose $A \in_{\mathcal{U}} \{0,1\}^{n \times l}$ (ie. A is an $(n \times l)$ matrix)
Set $T_j = A \bullet [j]$ where $[j]$ is the column l-vector with binary representation of j

Claim 2 *Let* $Y_i = \frac{1}{m}\sum_{j=1}^{m} \overline{B(e_i \oplus T_j)(x \bullet T_j)}$
Then $|Y_i - \mu_i| < \frac{1}{\delta}$ *with probability* $> 1 - \frac{1}{2n}$

Proof: By pairwise independent sampling,

$$\Pr[|Y_i - \mu_i| \geq \frac{1}{\delta}] \leq \frac{\sigma^2}{\delta^2 m} \leq \frac{\sigma^2}{\delta^2 \frac{2n}{\delta^2}} = \frac{\sigma^2}{2n} \leq \frac{1}{2n}$$

The last inequality is true since $\sigma^2 \leq 1$ by a previous homework assignment. □

Let $bit(Y_i) = \begin{cases} 1 \text{ if } Y_i < 0 \\ 0 \text{ if } Y_i \geq 0 \end{cases}$ This is the value of x_i predicted by Y_i.

Claim 3 *$bit(Y_i) = x_i$ for $i = 1, \ldots, n$ with probability $\geq \frac{1}{2}$*

Proof: By the previous claim, the probability that something goes wrong with the i^{th} bit is less than $\frac{1}{2n}$. Since there are n bits, the probability that something goes wrong with at least one bit is less than $\frac{n}{2n} = \frac{1}{2}$ and the claim follows. $\square$

The only problem is that Y_i seems to depend on x! So how do we actually compute Y_i without knowing x? Here's how: treat x as a row vector and T_j as a column vector. We then have

$$Y_i = \frac{1}{m} \sum_{j=1}^{m} \overline{B(e_i \oplus T_j)} \cdot \overline{(x \bullet A) \bullet [j]}$$

Notice that we don't need to know x, we just need to know $\mathcal{B} = x \bullet A \in \{0,1\}^l$. Now try all possible values of $\mathcal{B}$, with the correct value of $\mathcal{B}$ yielding the proper values for the Y_i's.

<u>Algorithm for S^B</u> (with input δ)

1. Let $m = \lceil 2n/\delta^2 \rceil$ and $l = \lceil log_2(m+1) \rceil$
2. Choose $A \in_{\mathcal{U}} \{0,1\}^{n \times l}$
3. Let $T_j = A \bullet [j]$ for $j = 1, \ldots, n$
4. For each $\mathcal{B} \in \{0,1\}^l$ and $i = 1, \ldots, n$
   Let $Y_i = \frac{1}{m} \sum_{j=1}^{m} \overline{B(e_i \oplus T_j)} \cdot \overline{\mathcal{B} \bullet [j]}$ and $x_i^{\mathcal{B}} = bit(Y_i^{\mathcal{B}})$
   Append $(x_1^{\mathcal{B}} \cdots x_n^{\mathcal{B}})$ to the output list

Finally, the runtime of S^B is

$$m + 2^l n T_B(n) \approx m \cdot n \cdot T_B(n) = \frac{2nT_B(n)}{\delta^2} = \frac{n^{O(1)} T_B(n)}{\delta^2}$$

Math 267a - Foundations of Cryptography

Lecture #18: 24 February 1997

Lecturer: Sam Buss

Scribe Notes by: Howard Skogman

36 Many Hidden Bits

Definition 36.1 *Let $r \geq 1 \in \mathbb{Z}$ ($r = r(n)$). Let*

$$f : \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$$

be a P-time function ensemble with $x \in \{0,1\}^n$ private, and $z \in \{0,1\}^{n \times r}$ public. z is a ($n \times r$ matrix over $\mathbb{Z}_2$. Let r inner product bits of $f(x)$ with respect to z is the vector $x \bullet z$

Definition 36.2 *The success probablility of an adversary $A : \{0,1\}^{l(n)} \times \{0,1\}^{n \times r} \longrightarrow \{0,1\}^r$ is defined by $\delta(n) = Pr_{X,Z}[f(A(f(X), Z)) = X \bullet Z] - 2^{-r}$. ($2^{-r}$ is the random guess probability)*

Definition 36.3 *The r-inner product bits are $S(n)$ secure if every adversary A has time-success ratio $\geq S(n)$. The r-inner product bits are hidden if they are n^c secure for all $c \geq 0$*

Theorem 36.4 *If there is a one way function then the r-inner product bits are hidden.*

Proof: We'll reduce to the case of one hidden bit i.e. we'll show that if the r-inner product bits are not hidden then 1-inner product bit is not hidden.

Let A be an adversary against the r-inner product bits. Let the success probability of A be denoted $\delta(n)$ and the run time be denoted $T(n)$. We'll construct an adversary S^A against a single inner product bit. [We know from a previous theorem that if f is a one-way function then its inner product bit is hidden]

The Algoithm for S^A is as follows:

Input $u = f(x) \in \{0,1\}^{l(n)}, y \in \{0,1\}^n$. The Goal is to output $x \bullet y$, one inner product bit.

Algorithm

1)Choose $i \in_U \{0,1\}^r - 0^r$.
2)Let l be the least value such that the lth bit of the binary representation of i is $\neq 0$ i.e $i_l = 1$.
3)Choose $z \in_U \{0,1\}^{n \times r}$
4)Modify column l of z to get the matrix z' such that $z' \bullet i = y$. z' is determined uniquely by z, y, i.
5)Output $A(u, z') \bullet i$

We hope this works since we hope that $A(U, Z') = A \bullet Z'$ and thus $A(U, Z') \bullet i = X \bullet (Z' \bullet i) = X \bullet Y$.

The Run time of $S^A = T(n) + n^{O(1)} \approx T(n)$. The success probability of S^A is

a. with probability $\geq \delta(n) + 2^{-r}$ $A(u, z') = x \bullet z'$ and the correct bit is output

b. if $A(u, z') \neq x \bullet z'$ then we have $A(u, z') \bullet i$ instead of $(x \bullet z') \bullet i$ and we hope this is correct half of the time.
Note that $A(u, z') \bullet i = (x \bullet z') \bullet i$ if and only if $[A(u, z') - (x \bullet z')] \bullet i = 0$. For $J \in \{0,1\}^r$, (we allow $J = 0^r$) $Pr_J[A(u, z') - x \bullet z') \bullet J = 0] = 1/2$, so $Pr_{I \in \{0,1\}^r - 0^r}[(A(u, z') - x \bullet z')\ bulletI = 0] = (2^{r-1} - 1)/(2^r - 1)$.

From a), b) the probability that S^A is correct is

$$\geq (\delta(n) + 2^{-r}) \times 1 + (1 - (\delta(n) + 2^{-r}))[(2^{r-1} - 1)/(2^r - 1)] = 1/2 + \delta(n)/2$$

.

So, the success probability of S^A is 2[Pr(A outputs the correct answer) - 1/2] $\geq \delta(n)$. So S^A has the same success probability as A. So the time success probability of S^A is the time success ratio of A.

QED.

$\square$

Example 36.5 For some concrete one way functions.

The low order i.e. log(log(n)) bits of the function $X \mapsto X^2 \pmod{pq}$ are hidden.

The low order bits i.e. log(log(n)) bits of RSA function are hidden asuuming the RSA is one-way.

In both cases these bits can be used to generate pseudorandom values by iterating the one-way functions.

Math 267a - Foundations of Cryptography

Lecture #19: 26 February 1997

Lecturer: Sam Buss

Scribe Notes by: Jeremy Martin

37 Statistical Distinguishability of Distributions

Definition: Let $\mathcal{D}_n$ and $\mathcal{E}_n$ be probability distributions on $\{0,1\}^n$. The *statistical distance* between $\mathcal{D}_n$ and $\mathcal{E}_n$ is

$$\mathrm{dist}(\mathcal{D}_n, \mathcal{E}_n) = \frac{1}{2} \sum_{\alpha \in \{0,1\}^n} \left| \Pr_X[X = \alpha] - \Pr_Y[Y = \alpha] \right|$$

where $X \in_{\mathcal{D}_n} \{0,1\}^n$ and $Y \in_{\mathcal{E}_n} \{0,1\}^n$. Equivalently,

$$\mathrm{dist}(\mathcal{D}_n, \mathcal{E}_n) = \max_{S \subset \{0,1\}^n} \left(\Pr_X[X \in S] - \Pr_Y[Y \in S] \right)$$

for the same X and Y. A third equivalent formula is the following. Let $t : \{0,1\}^n \to \{0,1\}$ and put $\delta_t(\mathcal{D}_n, \mathcal{E}_n) = |\Pr_X[t(X) = 1] - \Pr_Y[t(Y) = 1]|$. Then

$$\mathrm{dist}(\mathcal{D}_n, \mathcal{E}_n) = \max_t \delta_t(\mathcal{D}_n, \mathcal{E}_n)$$

Homework 17: Prove that these definitions are equivalent. Also, prove that the statistical distance is a metric.

Definition: $\mathcal{D}_n$ and $\mathcal{E}_n$ are *ϵ-statistically indistinguishable* iff $\mathrm{dist}(\mathcal{D}_n, \mathcal{E}_n) \leq \epsilon$.

38 Computational Indistinguishability of Distributions

Let $\mathcal{D}_n$,$\mathcal{E}_n$ be P-samplable distributions on $\{0,1\}^{l(n)}$, i.e.,

$$\mathcal{D}_n : \{0,1\}^{r(n)} \to \{0,1\}^{l(n)}$$
$$\mathcal{E}_n : \{0,1\}^{s(n)} \to \{0,1\}^{l(n)}$$

are P-time computable. Put $\mathcal{D}_n = D_n(\mathcal{U}_{r(n)})$ and $\mathcal{E}_n = E_n(\mathcal{U}_{s(n)})$, where $\mathcal{U}_{r(n)}$ and $\mathcal{U}_{s(n)}$ are uniformly chosen random variables of the appropriate length. An adversary for distinguishing $\mathcal{D}_n$ and $\mathcal{E}_n$, $A : \{0,1\}^{l(n)} \to \{0,1\}$, has success rate

$$\delta_n = \left| \Pr_X[A(X) = 1] - \Pr_Y[A(Y) = 1] \right|$$

where again $X \in_{\mathcal{D}_n} \{0,1\}^n$ and $Y \in_{\mathcal{E}_n} \{0,1\}^n$.

Definition: $\mathcal{D}_n$ and $\mathcal{E}_n$ are $S(n)$*-secure computationally indistinguishable* iff every adversary A has time-success ration $\geq S(n)$, and are *computationally indistinguishable* iff they are n^c-computationally indistinguishable for every $c > 0$.

Remark: If $\mathcal{D}_n$ and $\mathcal{E}_n$ are ϵ-statistically indistinguishable, then $\delta_n \leq \epsilon$ and $T(n) \geq 1$, so $\mathcal{D}_n$ and $\mathcal{E}_n$ are $1/\epsilon$-computationally indistinguishable.

Example: Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a PRNG. Then $f(\mathcal{U}_n)$ and $\mathcal{U}_{l(n)}$ are computationally indistinguishable. This is immediate from the definitions.

39 Strengthening the Hidden-Bit Theorems

Theorem: Let $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ be a one-way function. Let $Z \in_{\mathcal{U}} \{0,1\}^{n\times r}$ and $B \in_{\mathcal{U}} \{0,1\}^r$. Then the distributions

$$\mathcal{D}_n = \langle f(X), X \cdot Z, Z\rangle \quad \text{and} \quad \mathcal{E}_n = \langle f(X), B, Z\rangle$$

are computationally indistinguishable, provided $r = O(\log n)$.

Proof: Suppose A is an adversary distinguishing $\mathcal{D}_n$ and $\mathcal{E}_n$ with success rate $\mathcal{D}_n$ and runtime $T(n)$. We will construct an adversary S^A for determining the r-inner product bits of f.

S^A takes as input $Y = f(X)$ and $Z \in \{0,1\}^{n\times r}$; its goal is to output $X \cdot Z$. The algorithm proceeds as follows:

1. Choose $U, V \in_{\mathcal{U}} \{0,1\}^r$.
2. If $A(Y, U, Z) = 1$ then output U; otherwise output V.

The runtime of S^A is approximately equal to $T(n)$, and the success probability is

$$\begin{aligned}
&\Pr_{X,Z}[S^A(f(X), Z) \text{ outputs the correct answer}] \\
&\quad = \Pr[A(f(X), U, Z) = 1] \cdot \Pr[U = X \cdot Z \mid A(f(X), U, Z) = 1] \\
&\qquad + (1 - \underbrace{\Pr[A(f(X), U, Z) = 1]}_{\beta}) \cdot \Pr[V = X \cdot Z]
\end{aligned}$$

By Bayes' Formula, this becomes

$$\Pr[U = X \cdot Z] \cdot \Pr[A(f(X), U, Z) = 1 \mid U = X \cdot Z] + (1 - \beta) \cdot 2^{-r}$$

$$= 2^{-r} \cdot \underbrace{\Pr[A(f(X), X \cdot Z, Z) = 1]}_{\alpha} \; + \; (1 - \beta) \cdot 2^{-r}$$

By definition of δ we can assume w.o.l.o.g. $\alpha - \beta > \delta(n)$, so the last formula reduces to

$$2^{-r}(\alpha + 1 - \beta) \geq 2^{-r}(\delta(n) + 1)$$

Therefore the success rate of S^A is $\geq \delta(n)/2^r$, and the time-success ratio is $\leq 2^r \cdot T(n)/\delta(n) = n^{O(1)} \cdot T(n)/\delta(n)$ since $r = O(\log n)$. This makes S^A too strong an adversary against inner-product bits, a contradiction.

40 A Version of the Triangle Inequality

Theorem: Let D_n^1, D_n^2, D_n^3 be probability distributions on $\{0,1\}^n$. Let D_n^1 and D_n^2 be $S_{12}(n)$-computationally indistinguishable and let D_n^2 and D_n^3 be $S_{23}(n)$-computationally indistinguishable. Let

$$S_{13} = \frac{1}{\frac{1}{S_{12}} + \frac{1}{S_{23}}} \quad \left(\geq \frac{\min(S_{12}, S_{23})}{2} \right)$$

Then D_n^1 and D_n^3 are $S_{13}(n)$-computationally indistinguishable.

Homework 19: Prove this theorem.

Corollary: If the pairs D_n^1, D_n^2 and D_n^2, D_n^3 are computationally indistinguishable, then so is the pair D_n^1, D_n^3.

Theorem: Let $\mathcal{D}_n$ and $\mathcal{E}_n$ be computationally indistinguishable and P-samplable. Let $k(n)$ be a polynomial and

$$\mathcal{D}_n^{k(n)} = \underbrace{\mathcal{D}_n \times \mathcal{D}_n \times \cdots \times \mathcal{D}_n}_{k(n) \text{ copies}}$$

Define $\mathcal{E}_n^{k(n)}$ likewise. Then $\mathcal{D}_n^{k(n)}$ and $\mathcal{E}_n^{k(n)}$ are computationally indistinguishable.

Homework 20: Prove this theorem. (Hint: Recall the proof of the stretching theorem for PRNG's.)

41 Entropy and Information

Let $\mathcal{D}_n$ be a probability distribution on $\{0,1\}^n$ and $X \in_{\mathcal{D}_n} \{0,1\}^n$. We want to answer the following question: if we receive a sequence of values for X, how much information (approximately mesaured in bits) is conveyed by a particular value for X? Intuitively, if $\mathcal{D}_n \neq \mathcal{U}_n$, then we receive more information from *less likely* events. Also, we want to use shorter strings to represent more likely events.

Example: If you hear on the radio that traffic is flowing smoothly on the I-5, you don't learn very much because this is the normal situation. On the other hand, you receive quite a lot more information from hearing that a truck has overturned and there's cream of mushroom soup all over the freeway. Also, the radio announcer is likely to spend more time discussing the spilled soup than she would spend if there had been no accident.

Definition: Let $\alpha \in \{0,1\}^n$ be a possible value for X. Then

$$\mathrm{inform}_{\mathcal{D}_n}(\alpha) = -\log \left(\Pr_X[X = \alpha] \right)$$

where the logarithm is taken with base 2.

Math 267a - Foundations of Cryptography

Lecture #20: 28 February 1997

Lecturer: Sam Buss

Scribe Notes by: Anand Desai

42 Information and Entropy

Information in the sense of resolving uncertainty.
Applications: Design communication systems which optimize channel capacity/bandwidth.

Example 42.1 Morse Code
Language with 3 symbols: dot, dash, pause.
Design principle used by Morse - *"shorter codes for more common symbols"*

Example 42.2 Spoken English
Design principle observed - *"more commonly used words tend to be shorter"*

Let $\mathcal{D}_n$ be a distribution on $\{0,1\}^n$ where $\{0,1\}$ is the set of 'symbols'. Let $X \in_{\mathcal{D}_n} \{0,1\}^n$
Notation: Logarithms used in this lecture are always base two unless explicitly indicated otherwise.

Definition 42.3 *Shannon information of* $\alpha \in \{0,1\}^n$

$$\begin{aligned} inform_{\mathcal{D}_n}(\alpha) &= \log\left(\frac{1}{\Pr_X[X=\alpha]}\right) \\ &= -\log(\Pr_X[X=\alpha]) \end{aligned}$$

Definition 42.4 *The entropy of* $\mathcal{D}_n$ *is*

$$\begin{aligned} ent(\mathcal{D}_n) &= E_X[inform_{\mathcal{D}_n}(X)] \\ &= \sum_{\alpha\in\{0,1\}^n} \Pr_X[X=\alpha]\cdot\log\left(\frac{1}{\Pr_X[X=\alpha]}\right) \\ &= -\sum_{\alpha\in\{0,1\}^n} \Pr_X[X=\alpha]\cdot\log(\Pr_X[X=\alpha]) \end{aligned}$$

$ent(X) = ent(\mathcal{D}_n)$
Convention: $0 \cdot \log\left(\frac{1}{0}\right) = 0 \cdot \log 0 = 0$

Example 42.5

$$ent(\mathcal{U}_n) \;=\; 2^n \cdot \frac{1}{2^n} \cdot \log\left(\frac{1}{\frac{1}{2^n}}\right) = n$$

Example 42.6
Let $Pr[X = 0^n] = \frac{1}{2}$ and $Pr[X = 1u] = \frac{1}{2^n}$ for $u \in \{0,1\}^{n-1}$

$$\begin{aligned} ent(X) &= \frac{1}{2}\log 2 + 2^{n-1}\left(\frac{1}{2^n}\right)\log 2^n \\ &= \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot n \\ &= \frac{n+1}{2} \end{aligned}$$

Definition 42.7 *Let* $f : \{0,1\}^n \to \{0,1\}^*$ *where* $\{0,1\}^* = \bigcup_{i\geq 0}\{0,1\}^i$
We say f *is* prefix-free *iff for all* $x, y \in \{0,1\}^n$ $x \neq y$, $f(x)$ *is not a prefix of* $f(y)$.

We look for prefix-free encodings to transmit symbols. In the last example the encoding $f(0^n) = 0$ and $f(1u) = 1u$ would work well.

Theorem 42.8 $ent(\mathcal{D}_n) \leq n$
Proof:

$$\begin{aligned} ent(\mathcal{D}_n) - n &= \sum_\alpha \Pr[X=\alpha]\cdot(-\log(\Pr[X=\alpha]) - n) \\ &= \sum_\alpha \Pr[X=\alpha]\cdot \log\left(\frac{1}{\Pr[X=\alpha]\cdot 2^n}\right) \\ &\leq (\log e)\sum_\alpha \Pr[X=\alpha]\cdot\left(\frac{1}{\Pr[X=\alpha]\cdot 2^n} - 1\right) \\ &\leq (\log e)\sum_\alpha\left(\frac{1}{2^n} - \Pr[X=\alpha]\right) \\ &\leq (\log e)\cdot(1-1) = 0 \end{aligned}$$

We use the inequality $\log(E) \leq (\log e)(E-1)$ *above.* □

Homework 20. Suppose that X and Y are independent random variables and let $Z = \langle X, Y\rangle$.

$$ent(Z) = ent(X) + ent(Y)$$

Definition 42.9 (Information Divergence) *Let X and Y be random variables on $\{0,1\}^n$. The* information divergence *of Y w.r.t. X is*

$$\sum_{\alpha \in \{0,1\}^n} \Pr_X[X=\alpha] \cdot \log\left(\frac{\Pr_X[X=\alpha]}{\Pr_Y[Y=\alpha]}\right)$$

This is equal to,

$$\sum_{\alpha} \Pr_X[X=\alpha] \cdot (inform_Y(\alpha) - inform_X(\alpha))$$

Theorem 42.10 (Kullback - Liebler Inequality) *The information divergence of Y w.r.t. X is ≥ 0*

Proof:
The information divergence of Y w.r.t. X is

$$\begin{aligned}
&= -\sum_{\alpha \in \{0,1\}^n} \Pr_X[X=\alpha] \cdot \log\left(\frac{\Pr_Y[Y=\alpha]}{\Pr_X[X=\alpha]}\right) \\
&\geq -(\log e) \sum_{\alpha \in \{0,1\}^n} \Pr_X[X=\alpha] \cdot \left(\frac{\Pr_Y[Y=\alpha]}{\Pr_X[X=\alpha]} - 1\right) \\
&\geq -(\log e) \sum_{\alpha \in \{0,1\}^n} (\Pr_Y[Y=\alpha] - \Pr_X[X=\alpha]) \\
&\geq -\log e \cdot (1-1) = 0
\end{aligned}$$

□

Theorem 42.11 *Let X and Y be (not necessarily independent) random variables. Let $Z=(X,Y)$*

$$ent(Z) \leq ent(X) + ent(Y)$$

Proof: Let X' and Y' be independent random variables with the same distributions as X and Y respectively.

$$
\begin{aligned}
ent(Z) &= -\sum_{\alpha,\beta} Pr[X=\alpha \wedge Y=\beta] \cdot \log\left(Pr[X=\alpha \wedge Y=\beta]\right) \\
&\leq -\sum_{\alpha,\beta} Pr[X=\alpha \wedge Y=\beta] \cdot \log(Pr[X'=\alpha \wedge Y'=\beta]) \quad \text{-by Kullback-Liebler inequality} \\
&\leq -\sum_{\alpha,\beta} Pr[X=\alpha \wedge Y=\beta] \cdot \log\left(Pr[X=\alpha] \cdot \Pr[Y=\beta]\right) \\
&\leq -\sum_{\alpha,\beta} Pr[X=\alpha \wedge Y=\beta] \cdot (\log(Pr[X=\alpha]) + \log(\Pr[Y=\beta])) \\
&\leq -\sum_{\alpha,\beta} Pr[Y=\beta|X=\alpha] \cdot \Pr[X=\alpha] \cdot \log(\Pr[X=\alpha]) \\
&\quad -\sum_{\alpha,\beta} Pr[X=\alpha|Y=\beta] \cdot \Pr[Y=\beta] \cdot \log(\Pr[Y=\beta]) \\
&\leq -\sum_{\alpha} \left(\sum_{\beta} Pr[Y=\beta|X=\alpha]\right) \cdot \Pr[X=\alpha] \cdot \log(\Pr[X=\alpha]) \\
&\quad -\sum_{\beta} \left(\sum_{\alpha} Pr[X=\alpha|Y=\beta]\right) \cdot \Pr[Y=\beta] \cdot \log(\Pr[Y=\beta]) \\
&\leq 1 \cdot ent(X) + 1 \cdot ent(Y) \\
&\leq ent(X) + ent(Y)
\end{aligned}
$$

$\square$

Math 267a - Foundations of Cryptography

Lecture #21: 3 March 1997

Lecturer: Sam Buss

Scribe Notes by: Jennifer Wagner

43 Prefix-free codes

Recall: A prefix-free code is a function $f : \{0,1\}^n \longrightarrow \{0,1\}^*$ such that if $x \neq y$, then $f(x)$ is not a prefix of $f(y)$. Our measure of efficiency of a prefix-free code is $E_X[|f(X)|]$ where $|f(X)|$ is the number of bits of $f(X)$. The lower this number is, the better.

Theorem 43.1 *(Kraft inequality) Let f be prefix-free. Then*

$$\sum_{x \in \{0,1\}^n} 2^{-|f(x)|} \leq 1.$$

Proof: Let $l = \max\{|f(x)| : x \in \{0,1\}^n\}$. For each x, consider the set of $y \in \{0,1\}^l$ such that $f(x)$ is a prefix of y. This set is $\frac{1}{2^{|f(x)|}}$ of the set $\{0,1\}^l$. Moreover, for distinct x the sets are disjoint. Thus summing $\frac{1}{2^{|f(x)|}}$ over all x gives a number no larger than 1. □

Theorem 43.2 *Let f be prefix-free, and $X \in_{\mathcal{D}} \{0,1\}^n$ where $\mathcal{D}$ is some probability distribution. Then*

$$E_X[|f(X)|] \geq \text{ent}(X).$$

Proof: Let $Y \in_{\mathcal{E}} \{0,1\}^n \cup \{\lambda\}$ be such that $\text{Pr}_Y[Y = \alpha] = 2^{-|f(\alpha)|}$ for $\alpha \in \{0,1\}^n$. Then the Kraft inequality gives that

$$\text{Pr}_Y[Y = \lambda] = 1 - \sum_{\alpha} 2^{-|f(\alpha)|} \geq 0.$$

We want to show that $E_X[|f(X)|] - \text{ent}(X) \geq 0$. But

$$\begin{aligned} E_X[|f(X)|] - \text{ent}(X) &= \sum_{\alpha} \text{Pr}_X[X = \alpha](|f(\alpha)| + \log(\text{Pr}_X[X = \alpha])) \\ &= \sum_{\alpha} \text{Pr}_X[X = \alpha] \log(\frac{\text{Pr}_X[X = \alpha]}{2^{-|f(\alpha)|}}) \\ &= \sum_{\alpha} \text{Pr}_X[X = \alpha] \log(\frac{\text{Pr}_X[X = \alpha]}{\text{Pr}_Y[Y = \alpha]}). \end{aligned}$$

But this last sum is ≥ 0 by the Kullback-Liebler theorem. □

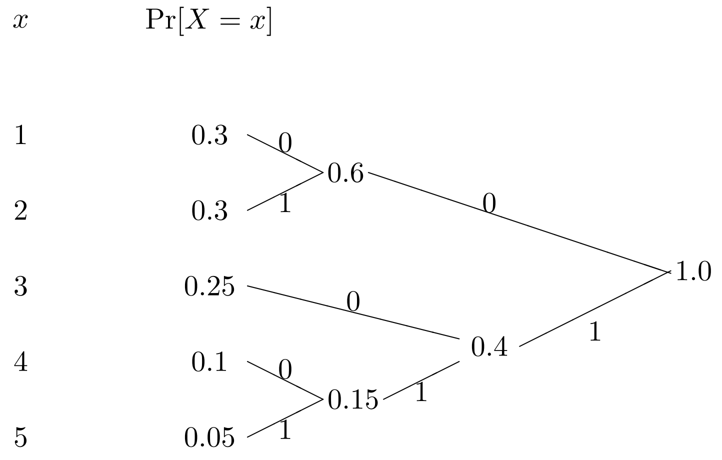
44 Huffman codes

Intuitively, one might hope for a prefix-free code with $E_X[|f(X)|] \leq \text{ent}(X) + 1$, which would put the expected number of bits close to within rounding error of the entropy. This is achieved by *Huffman codes.*

Set-up: Consider a finite set $S = \{S_1, S_2, ...S_m\}$ with $X \in S$ a random variable. Let $p_1, p_2, ..., p_m$ be $p_i = \Pr_X[X = S_i]$ where $p_1 \geq p_2 \geq ... \geq p_m > 0$. We will define the Huffman encoding function f inductively. For $m = 2$, let $f(S_1) = 0$ and $f(S_2) = 1$. If $m \geq 3$, form a set $S' = \{S_1, S_2, ..., S_{m-2}, \{S_{m-1}S_m\}\}$ by combining S_{m-1} and S_m into a single element. Let $p'_i = p_i = \Pr_Y[Y = S'_i]$ if $1 \leq i < m - 1$, and $p'_{m-1} = p_{m-1} + p_m = \Pr_Y[Y = S'_{m-1}]$. Note that S' may need to be reordered to achieve $p'_1 \geq p'_2 \geq ... \geq p'_{m-1}$. Inductively, find the Huffman encoding function f' on S'. Then define the Huffman encoding function f on S by

$$\begin{aligned} f(S_i) &= f'(S_i) \text{ if } 1 \leq i < m - 1 \\ f(S_{m-1}) &= f'(\{S_{m-1}S_m\})0 \\ f(S_m) &= f'(\{S_{m-1}S_m\})1. \end{aligned}$$

Example 44.1 Let $S = \{1, 2, 3, 4, 5\}$ with the probabilities given in the table below. Form the tree by iterating the process of combining the two lowest probabilities.



The Huffman code is then found by reading the tree from the root to the appropriate leaf. Thus $f(1) = 00, f(2) = 01, f(3) = 10, f(4) = 110, f(5) = 111$. This is a prefix-free code by design. In addition, it assigns the longest codes to the least common events.

Theorem 44.2 *If f is a Huffman coding of a probability distribution on $\{0, 1\}^n$, then* $E_X[|f(X)|] \leq \text{ent}(X) + 1$.

Homework 21. Prove the previous theorem. Hint: Suppose more than one event has nonzero probability. Show that $E_X[|f(X)|] \leq \mathrm{ent}(X) + 1 - 2 \cdot \min_\alpha\{\Pr_X[X = \alpha]\}$. Use induction on the construction of the Huffman encoding.

Homework 22 Show that the Huffman codes are optimal prefix-free codes.

An alternative to the hint for the first homework above, is that one can first show that there is a prefix-free encoding which has the property that $|f(x)| = \lceil ent(x) \rceil$ for all x. This enoding has the property that $E_X[|f(X)|] \leq \mathrm{ent}(X) + 1$. Finally, one can use the second homework to show that Huffman encodings also have this property.

Math 267a - Foundations of Cryptogaphy

Lecture #22: 5 March 1997

Lecturer: Sam Buss

Scribe Notes by: Robert Ellis

45 PRNG's from One-Way Functions

Recall that a pseudorandom number generator (PRNG) can be constructed from a one-way permutation by adding hidden bits on the end of the outputs of the one-way function. For this, it has been seen that bijectiveness of the one-way function is crucial. A PRNG can also be constructed from a one-way function, via pseudoentropy and false entropy generators. Here these new generators are described, followed by an overview of the construction of a PRNG from a one-way function.

Definition: A function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a $S(n)$-*secure pseudoentropy generator* with pseudoentropy $p(n)$ provided $f(\mathcal{U}_n)$ has $S(n)$-secure pseudoentropy $\geq n + p(n)$.

Recall that $\mathcal{U}_n$ is the uniform distribution, which has entropy n. It is a general fact that if h is a function, and a distribution $\mathcal{D}$ has entropy $\mathrm{ent}(\mathcal{D})$, then $h(\mathcal{D})$ has entropy $\leq \mathrm{ent}(\mathcal{D})$. Therefore $f(\mathcal{U}_n)$ cannot have true entropy $n + p(n)$; however $f(\mathcal{U}_n)$ may be indistinguishable from a distribution on $\{0,1\}^{l(n)}$ with entropy $\geq n + p(n)$ - in this case f is a pseudoentropy generator.

Definition: A function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ is a $S(n)$-secure false entropy generator with false entropy $p(n)$ provided $f(\mathcal{U}_n)$ had $S(n)$-secure pseudoentropy $\geq \mathrm{ent}(f(\mathcal{U}_n)) + p(n)$.

Since $\mathrm{ent}(f(\mathcal{U}_n)) \leq \mathrm{ent}(\mathcal{U}_n)$, It follows from the definitions that a pseudoentropy generator is also a false entropy generator. Now we outline a construction by Håstad, Impagliazzo, Levin, and Luby of a PRNG from a one-way function.

$$\begin{aligned} \textbf{one-way function} &\Rightarrow \textbf{false entropy generator} \\ &\Rightarrow \textbf{pseudoentropy generator} \\ &\Rightarrow \textbf{pseudorandom number generator} \end{aligned}$$

Our outline will combine the last two steps, and produce a PRNG from a false entropy generator.

(one-way function $\Rightarrow$ false entropy generator)

Let f be a one-way function ($x \mapsto f(x)$). Then we may construct a function g for which $g(x, z) = \langle f(x), x \bullet z, z \rangle$, which is indistinguishable from $\langle f(x), b, z \rangle$, where b consists of "random" bits. (Here, z is a matrix of appropriate size for the dot product.)

Now, we can break z by columns into z_1 and z_2, alter the function g so that $g(x, z_1, z_2) = \langle f(x), x \bullet z_1, x \bullet z_2, z_1, z_2 \rangle$, and by choosing the correct splitting of z, we nearly have injectivity for the mapping $(x, z_1) \mapsto \langle f(x), x \bullet z, z_1 \rangle$. This approximation to one-to-one behavior and the extra hidden bits (almost random bits) give the false entropy.

(false entropy generator ⇒ pseudorandom number generator)

Let g be a false entropy generator, with $x \mapsto g(x)$, where we may think of x as being the (x, z_1, z_2) in the previous paragraph. Now construct the function

$$h(x, z_1, z_2) = \left\langle g^{k(n)} \bullet z_1, x \bullet z_2, z_1, z_2 \right\rangle,$$

where z_1, and z_2 are matrices of appropriate size; furthermore,

$$\begin{aligned} \overline{x} &= x_1 x_2 \cdots x_{k(n)}, \quad \text{and} \\ g^{k(n)}(\overline{x}) &= g(x_1) g(x_2) \cdots g(x_n), \end{aligned}$$

and z_1 is large enough to obscure the values of g. The claim which will be left unproven here is that h is a PRNG.

From a previous homework, we already have that a one-way function can be constructed from a PRNG; thus one-way functions exist ⇔ PRNG's exist. Note that we have candidates for each type, but can't prove their respective memberships.

46 Hash Functions and One-Way Hash Functions

Definition: Let $h_y(x) = h(y, x)$, and let $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$, and $X \in_{\mathcal{U}} \{0,1\}^n$. A function $h : \{0,1\}^{l(n)} \times \{0,1\}^n \to \{0,1\}^{m(n)}$ (where in general, $m(n) < n$) is a *Universal Hash Function* provided:

1. For all $x \in \{0,1\}^n$, and for all $\alpha \in \{0,1\}^{m(n)}$,

$$\Pr_Y[h_Y(x) = \alpha] = \frac{1}{2^{m(n)}}.$$

(We can think of this condition as the hash function sending domain elements to all range elements with equal probability.)

2. For all x_1, $x_2 \in_{\mathcal{U}} \{0,1\}^n$, and for all α_1, $\alpha_2 \in \{0,1\}^{m(n)}$, where $x_1 \neq x_2$,

$$\Pr_Y[h_Y(x_1) = \alpha_1 \wedge h_Y(x_2) = \alpha_2] = \frac{1}{2^{2m(n)}}.$$

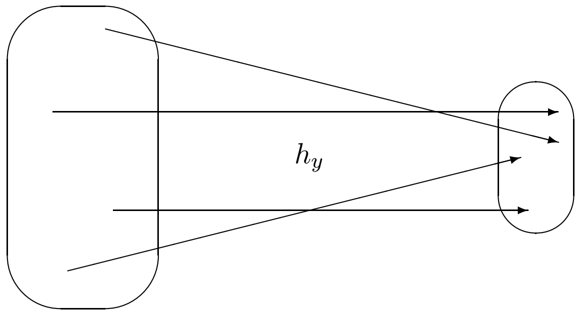
(We can think of this condition as pairwise independence of the hashings of x_1 and x_2.)

Intuitively, a hash function maps a large set to a small set in an essentially random fashion; however, it is usually desired to have the portion of the domain that is actually used at any given point in time to be relatively small compared to the range set.

Example. Consider the generalized inner product. View x as a row n-vector, and y as a $(n+1) \times m(n)$ matrix. In the language of the definition we have $l(n) = (n+1) \cdot m(n)$. Let $h_y(x) = \langle x, 1 \rangle \cdot y$. Then h is a universal hash function.

Definition: A hashing *collision* occurs if $x_1 \neq x_2$ and $h_y(x_1) = h_y(x_2)$.

What follows now is a loose definition of one-way hash functions and universal one-way hash functions; one-way hash functions must be resistant to both attacks against their one-way character and to collisions.



$\{0,1\}^n$ $\{0,1\}^{m(n)}$

Figure 1: Representation of a hash function, h with parameter $y \in \{0,1\}^{l(n)}$.

Definition: A *(universal) one-way hash function*, h, is a (universal) hash function which is secure against the following types of attacks:

1. the "one-wayness" attack:
   Let $X \in_{\mathcal{U}} \{0,1\}^n$, $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$. Then h must be secure against an adversary A against its one-way character; i.e. we require

   $$\Pr_{X,Y}[h_Y(A(h_Y(X),Y)) = h_Y(X)]$$

   to be sufficiently small, where the parameter Y is publicly known.

2. the "birthday" attack:
   Let $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$. Given an adversary A against collisions of the hash function, we must have the probability that an adversary A' discovers a collision; i.e.,

   $$\Pr_Y[A'(Y) = \langle x_1, x_2\rangle : x_1 \neq x_2 \text{ and } h_y(x_1) = h_y(x_2)]$$

   is sufficiently small for the desired security parameter (usually $S(n)$).

Applications: Let $h : \{0,1\}^* \to \{0,1\}^n$ be a hash function.

1. message verification or message digest:
   Alice, sending a message m, computes the hash value $\sigma = h(m)$ and sends both. Bob can tell from the pair (m, σ) whether m has been corrupted, assuming σ has not been corrupted.

2. message registration:
   Alice wants to keep m private but later be able to prove she knew m. Alice computes $\sigma = h(m)$ and publishes σ, perhaps in a classified ad in the New York Times. Later anyone given m can verify that $h(m) = \sigma$ and Alice knew m. An example of this is the patent application process, where it is desired to keep the patent information secret, while giving proof of knowledge of the patent information, for instance by using a standard widely-accepted hash function.

Math 267a - Foundations of Cryptography

Lecture #23: 12 March 1997

Lecturer: Sam Buss

Scribe Notes by: Tin Yen Lee

47 Applications of Hash Functions

Let $h : \{0,1\}^* \longrightarrow \{0,1\}^n$ be a hash function (not 1-1 but hard to invert)

Recall: Attacks on Hash Functions

1. Passive attacks: Alice chooses the message m and computes its hash value $h(m)$. Eve only gets $h(m)$ and tries to find m' such that $h(m') = h(m)$.

2. Birthday attacks: Eve tries to find a general m and m' such that $m \neq m'$ and $h(m) = h(m')$. One use for this would be that Eve could sign a message m and later claim that she signed the message m' instead.

Application I: Message Verification
Alice sends Bob a message m and $\sigma = h(m)$, the hash function value. Bob can tell from (m, σ) that m has not been corrupted, assuming that σ has not been corrupted.

Application II: Message Registration
Alice needs to keep m private, but later needs to be able to prove that she knew m. Thus, Alice computes and publishes $\sigma = h(m)$ in the New York Times. Later, anyone, given m, can verify that $h(m) = \sigma$ and that Alice knew m before.

Application III: Message Signatures
Alice has a public key z and a private key x with the associated encryption/decryption function. Alice wants to send a message m to Bob in such a way that Bob is sure that it came from Alice. Alice computes $\sigma = E_x(h(m))$, and sends m and σ to Bob. (This transaction may be intercepted by Eve.) Bob, using the public key of Alice, computes $E_z(\sigma)$ and checks that this is equal to $h(m)$. Since σ has been computed with Alice's private key, Bob can be sure that Alice wrote the message.

Application IV: Verisign Application
Alice writes a program m for internet distribution. She computes $\sigma = E_x(h(m))$ and registers σ and the public key z with a trusted certificate manager. (ie. Verisign could be the certificate

manager and Microsoft could be Alice.) She posts σ and m on the internet. Bob downloads σ and m, and checks that $E_z(\sigma) = h(m)$, obtaining z from Verisign. He then can run m without fear of malicious programs.

48 Birthday Attack Success Probability

Eve is trying to find a general m and m' such that $m \neq m'$ and $h(m) = h(m')$. Eve might pick a random $x_1, x_2, x_3, \ldots$ until she finds a duplicate $h(x_i)$ value. What is her chance of success after picking $x_1, \ldots, x_k$?

Remark: For passive attacks, Eve, given σ, expects to try about $k = \frac{2^n}{2}$ many x_i's before she succeeds with non-negligible probability.

The birthday attack has a much better success probability. Let us assume that h is a random function.

$$\begin{aligned}
&\text{Prob[Eve fails]} \\
&\leq \text{Prob}[h(x_1) \neq h(x_2) \text{ and } h(x_3) \notin \{h(x_1), h(x_2)\} \\
&\qquad \text{and } h(x_4) \notin \{h(x_1), h(x_2), h(x_3)\} \\
&\qquad \text{and } \ldots \text{ and } h(x_k) \notin \{h(x_1), \ldots, h(x_{k-1})\}] \\
&\leq (1 - \frac{1}{2^n})(1 - \frac{2}{2^n})(1 - \frac{3}{2^n}) \ldots (1 - \frac{k-1}{2^n}) = \alpha
\end{aligned}$$

Now,

$$\begin{aligned}
\ln(\alpha) &= \ln(1 - \frac{1}{2^n}) + \ln(1 - \frac{2}{2^n}) + \ldots + \ln(1 - \frac{k-1}{2^n}) \\
&\leq -\frac{1}{2^n} - \frac{2}{2^n} - \ldots - \frac{k-1}{2^n} \qquad \text{since } \ln(x) \leq x - 1 \\
&= -\frac{(k-1)k}{2 \cdot 2^n}
\end{aligned}$$

Thus,

$$\ln(\alpha) \lessapprox -\frac{1}{2} \Longrightarrow \alpha \leq e^{-\frac{1}{2}} \approx .6065$$

So Eve would succeed with probability $\geq 1 - .607 = .393$ after $\sqrt{2^n} = 2^{n/2}$ trials. This is a lot less work than she would need for a passive attack.

It would be to Eve's advantage to do this because if Eve, for instance, generates good contracts $g_1, g_2, g_3, \ldots$ and bad contracts $b_1, b_2, b_3, \ldots$, until she finds a b_i and a g_j such that $h(b_i) = h(g_j)$, she could then ask Alice to sign g_j. Alice would then give back $E_x(h(g_j))$, but then Eve could convince Bob that Alice really signed b_i, since $E_x(h(b_i)) = E_x(h(g_j))$.

Math 267a - Foundations of Cryptography

Lecture #24: 14 March 1997

Lecturer: Sam Buss

Scribe Notes by: Sam Buss

49 Thwarting the birthday attack

The birthday attack described in the previous lecture can often be foiled by adding a random string as padding to the message before using the hash function.

As an example, consider the case where Alice and Bob are communicating with each other in order to sign a contract. To sign the contract, they will jointly compute the hash function applied to the message and then each will encrypt the hash function with their private key. However, Alice and Bob do not trust each other completely and each wants to be sure that the other one is not using a birthday attack. The protocol that Alice and Bob follows is:

1. Alice and Bob jointly agree on a contract c. We presume the contract is coded as a string of bits, so $c \in \{0,1\}^*$.

2. Alice chooses a random string $u_A \in \{0,1\}^p$ and Bob chooses a random string $u_B \in \{0,1\}^*$ where p is a modestly large integer. They reveal their random strings to each other, nearly simultaneously.

3. Alice and Bob concatenate these to form the message $m = cu_Au_B$. They compute $h(m)$ where h is a hash function.

4. Alice and Bob both sign the message: Alice computes $E_{x_A}(h(m))$ and Bob computes $E_{x_B}(h(m))$. They exchange these two values and use them as signatures. (Here x_A and x_B are the private keys of Alice and Bob.)

Alice feels secure that Bob was not able to secretly produce a corrupted, bad contract with the same hash value as the agreed upon contract, since Bob is unable to start a birthday attack until he knows Alice's random string u_A. (Note that it behooves Alice to require Bob to reveal his random string at the same time, or nearly the same time, as she reveals her, since otherwise Bob has a chance to mount a birthday attack after Alice reveals her key and before Bob reveals his.) The situation is symmetric for Bob, who likewise feels secure that Alice has not been able to mount a birthday attack against him.

50 Blinded signatures

50.1 The dangers of signing gibberish (that is, random messages).

We now consider the sitation where Bob has a public and private key cryptosystem based on RSA. Recall that this means Bob has a private key $x = \langle p, q, d\rangle$ and a public key $z = \langle e, n\rangle$ where $pq = n$. Bob's private encryption/decryption function is

$$E_x(m) \;=\; m^d \bmod n.$$

and his public encryption/decryption function is

$$E_z(m) \;=\; m^e \bmod n.$$

In this section, we define Bob's signature of a message m to be the value $E_x(m)$; in particular, Bob is not signing the hash of a message, he is signing the actual message.

We now suppose that Bob is willing to sign gibberish messages: we'll demonstrate that this is as dangerous an act as signing a blank check or a blank sheet of paper. (Anecdote: in the previous day's paper was a story describing two accused murderers in Phildelphia who claimed that the police had induced them to sign blank sheets of paper and then typed confessions above their signature. Their trial ended on the day of this lecture with their acquital, reportedly due to lack of physical evidence connecting them to the crime.)

Returning the cryptographic situation, we suppose Eve has heard about Bob's willingness to sign gibberish messages. She then uses the following scheme to obtains Bob's signature on a message m of her choosing:

1. Eve chooses a random $k \in Z_m^*$. She forms $g = mk^e \bmod n$.
2. Eve asks Bob to sign g. Bob complies as it appears to be gibberish and returns the value $E_x(g)$ to Eve.
3. Eve then computes $k^{-1} \bmod n$ using Euclid's algorithm and the fact that n and k are relatively prime. She then computes

$$\begin{aligned} E_x(g)\cdot k^{-1} \bmod n &= m^d \cdot k^{ed} \cdot k^{-1} \bmod n \\ &= m^d \bmod n \\ &= E_x(m) \end{aligned}$$

The second equality above holds since the order of Z_n^* is $ed - 1$, so $k^{ed} \equiv k \bmod n$.

Eve now has the value of $E_x(m)$ which is message m signed using Bob's private key.

It remains to justify the fact that Bob was willing to sign the message $g = mk^e$. Since k was chosen uniformly at random from Z_n^*, then k^e is also uniformly randomly distributed from Z_n^*. Therefore, since $\gcd(m, n) = 1$, the value of g is independent of m and is uniformly distributed in Z_n^*. Therefore the message g, being completely random, will most likely consist of just "gibberish".

50.2 Blinded signatures

The above construction showed the danger of signing random messages or "gibberish". However, from a different point of view, we can think of g as being a *blinded* form of m: what we mean by this is that knowledge of g gives no information about m to anyone except Eve, since only Eve knows the value of k. This allows Bob to offer *blindfolded signatures* as a service: that is to say, Bob may wish to agree that he will sign any message m without being able to see the contents of message m.

To give a physical analogue, suppose Eve wants Bob to sign message m. We can think of her inserting the message m written on a piece of ordinary paper into an envelope along with a slip of carbon paper (this is analogous to her blinding the message by multiplying by k^e). She then lets Bob sign on the outside of the envelope with a stylus; the carbon paper transfers the signature to the blinded message inside (this is analogous to Bob signing the blinded massage). Then, away from the presence of Bob, Eve can remove the message from the envelope (this corresponds to unblinding by multiplying by k). In this way, Bob has signed a message, but has no information about what he has signed.

A number of plausible and semi-plausible uses for blinded signatures have been proposed. Some of these involve the use of "digital coins" which function with anonymity similar to cash. (Remark: many of these schemes seem quite dangerous, in that they allow way too much freedom for activities such as money laundering or extortion.) We'll present a simple application that allows voters to anonymously donote money to a politician, without their bank knowing to whom they donated.

The scenario is as follows. Bob is a bank, and has a particular set of public and private keys z and x, which he uses for the following purpose: he announces that he is willing to sign any blinded message with his private key and, furthermore, if anyone presents him with a string of ascii symbols which have been encoded with his private key, then he will pay that person \$1000. A number of voters such as Alice then randomly choose a longish string of ascii characters, blind them and present them to Bob for signature. Bob charges them \$1000 plus a small service charge for this. Alice unblinds the signature, so that she now has a string of ascii symbols encoded with Bob's private key. She donates this to a politician (Jack, say). Jack takes the signed ascii string to Bob, who pays him \$1000 (minus a small service charge) after verifying that he has not already paid on that particular ascii string.

There are several assumptions in the above scenario. One is that ascii strings are quite rare compared to the set of all possible strings, so the probability is very low that someone who doesn't know Bob's private key can find a signature of Bob's on an ascii string. Second, Bob has to announce that this particular private/public key is solely for the purpose of signing \$1000 coins. Third, Alice has to choose her random string of ascii symbols to be long enough so that no one else will randomly choose the same string.

As for the privacy aspects of the above procedure. Bob knows that Alice obtained a \$1000 digital coin, but presumably many other voters did the same. When Bob received the unblinded version of the coin from Jack for payment, he has absolutely no information about which of the coins he is redeeming. That is to say, he has no way to link the action of giving Alice the blinded signature to the action of Jack returning the unblinded signature. So Bob has no way of knowing which voters donated to which candidates.

Math 267a - Foundations of Cryptography

Homework Problems

This is a complete list of the homework assignments. The problems are also stated in the preceding scribe notes, often with substantial hints.

1. Prove that, for all constants $0 < \delta < \frac{1}{2}$ and $c > 0$,
$$BPP(\frac{1}{n^c}) \;=\; BPP(\delta) \;=\; BPP(\frac{1}{2} - 2^{-n^c}).$$
2. Prove that if $P = PP$, then $P = NP$. In addition, if all predicates in PP are feasible, then all predicates in NP are feasible.
3. Prove that $BPP \subseteq P/poly$ and $RP \subseteq P/poly$.
4. Prove that if that $P = NP$, then pseudorandom number generators do not exist.
5. Prove that if $P = NP$ then NP-search problems do not exist.
6. Prove that if $P = NP$, then NP-search problems are solvable in polynomial time.
7. Prove that there is a polynomial time algorithm for finding square roots modulo a prime or a prime power.
8. Prove that a pseudo-random number generator is next-bit unpredictable.
9. Prove that if g is a pseudorandom number generator, then the stream cryptosystem constructed from g is secure against passive attacks where the adversary may force Alice to send one of two messages.
10. Prove that if g is a pseudorandom number generator, then the stream cryptosystem constructed from g is secure against simple plaintext attacks.
11. Formulate good notions of simple passive attacks and simple plaintext attacks block cryptosystems. Prove that if f is a pseudorandom function generator, then the block cryptosystem based on f is secure.
12. Generalize the linear polynomial space method for generating pairwise independent variables to a method for generating k-wise independent values. Analyze the amount of randomness used by this method.
13. Let X be a $\{-1, 1\}$-valued random variable. Let $p = Pr[X = 1]$. Show the following: (a) $E[X] = 2p - 1$, (b) $Var(X) = 4p(p - 1)$, and (c) $Var(X) \leq 1$.
14. State a prove a generalization of Markov's Inequality with $X \geq a$ (or $X \leq a$) instead of $X \geq 0$.

15. Re-do the proof that $RP(\frac{1}{n}) = RP(\frac{1}{2})$ using pairwise independent sampling. Compare the amount of randomness and the number of samples in the new proof to the earlier proof.

16. Give an example of a function which has hidden inner product bit, but which is not one-way.

17. Prove that the various characterizations of *statistical distance* are equivalent. Also, prove that statistical distance is a metric.

18. Prove that if $\mathcal{D}_n^1$ and $\mathcal{D}_n^2$ are $S_{12}(n)$-computationally indistinguishable and that if $\mathcal{D}_n^2$ and $\mathcal{D}_n^3$ are $S_{23}(n)$-computationally indistinguishable, then $\mathcal{D}_n^1$ and $\mathcal{D}_n^3$ are $S_{13}(n)$ computationally indistinguishable where
$$S_{13} = \frac{1}{1/S_{12} + 1/S_{23}}.$$

19. Let $k(n) = n^{O(1)}$. Prove that $\mathcal{D}_n^{k(n)}$ and $\mathcal{E}_n^{k(n)}$ are computationally indistinguishable, provided that $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable.

20. Prove directly that if X and Y are independent, then $ent(\langle X, Y\rangle) = ent(X) + ent(Y)$.

21. Prove that Huffman codings, f, have $E_X[|f(X)|] \leq ent(X) + 1$.

22. Prove that Huffman encodings are optimal prefix-free encodings.