RSA, the Chinese Remainder Theorem, and Remote Coin Flipping

CS70 Summer 2016 - Lecture 7B

David Dinh

02 August 2016

UC Berkeley

Agenda

RSA

The Chinese remainder theorem

Euler's Criterion

Blum's coin-flipping scheme *Slides marked with an asterisk* are considered enrichment material and will not be tested on the exam. Think of them as gigs.*

Encryption

Motivation

Let's say I'm trying to buy something on Amazon. Need to send Amazon my credit card number.

Motivation

Let's say I'm trying to buy something on Amazon. Need to send Amazon my credit card number.

Problem: what if somebody (let's say NSA) is snooping on my network connection?

Motivation

Let's say I'm trying to buy something on Amazon. Need to send Amazon my credit card number.

Problem: what if somebody (let's say NSA) is snooping on my network connection?

Goal: transmit my credit card number to Amazon without any eavesdroppers knowing what they are.

Private Key Crypto: One-Time Pad

Very simple way to encrypt messages.

Recall the truth table of xor (denoted $\oplus$): $A \oplus B = 1$ if and only if exactly one of A, B are 1.

Private Key Crypto: One-Time Pad

Very simple way to encrypt messages.

Recall the truth table of xor (denoted $\oplus$): $A \oplus B = 1$ if and only if exactly one of A, B are 1.

Simple encryption scheme ("one time pad"): given a *plaintext* we want to encrypt (e.g. credit card number, represented as a bitstring) and a *key* of equal length, xor each bit of the plaintext with the corresponding bit of the key to get a *ciphertext*.

Private Key Crypto: One-Time Pad

$x \oplus y = (x \vee y) \wedge \overline{(x \wedge y)}$

Very simple way to encrypt messages.

Recall the truth table of xor (denoted $\oplus$): $A \oplus B = 1$ if and only if exactly one of A, B are 1.

Simple encryption scheme (”one time pad”): given a *plaintext* we want to encrypt (e.g. credit card number, represented as a bitstring) and a *key* of equal length, xor each bit of the plaintext with the corresponding bit of the key to get a *ciphertext*.

How do we decrypt? Notice that $x \oplus y \oplus x = y \oplus x \oplus x = y \oplus 0 = y$. So: just xor the ciphertext with the key, bitwise, to get plaintext back.

CC: 1 0

OTP 1 1

Cipher 0 1 $\longrightarrow$ Amazon

1 0

1 1

0 1

Private Key Crypto: One-Time Pad

Very simple way to encrypt messages.

Recall the truth table of xor (denoted $\oplus$): $A \oplus B = 1$ if and only if exactly one of A, B are 1.

Simple encryption scheme ("one time pad"): given a *plaintext* we want to encrypt (e.g. credit card number, represented as a bitstring) and a *key* of equal length, xor each bit of the plaintext with the corresponding bit of the key to get a *ciphertext*.

How do we decrypt? Notice that $x \oplus y \oplus x = y \oplus x \oplus x = y \oplus 0 = y$. So: just xor the ciphertext with the key, bitwise, to get plaintext back.

Example: let's say my credit card has a bit representation of 01101. Pick key 11001. Ciphertext is 10100. Easy to verify that bitwise xor of 10100 and 11001 is 01101.

OTP: Pros and Cons

Why is OTP secure?

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

For every possible plaintext p, there exists a key k such that $c = p \oplus k$. Why?

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

of the same length

For every possible plaintext p, there exists a key k such that $c = p \oplus k$. Why? Just let $k = c \oplus p$.

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

For every possible plaintext p, there exists a key k such that $c = p \oplus k$. Why? Just let $k = c \oplus p$.

What's wrong with OTP?

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

For every possible plaintext p, there exists a key k such that $c = p \oplus k$. Why? Just let $k = c \oplus p$.

What's wrong with OTP?

- Need a really long key. Same length as input! Fine for credit card numbers, maybe not so fine for a few TB of top-secret blueprints for your next supervillain base...

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

For every possible plaintext p, there exists a key k such that $c = p \oplus k$. Why? Just let $k = c \oplus p$.

What's wrong with OTP?

- Need a really long key. Same length as input! Fine for credit card numbers, maybe not so fine for a few TB of top-secret blueprints for your next supervillain base...
- Can't reuse key twice without leaking info. Let's say I send $p_1 \oplus k$ and $p_2 \oplus k$. Then NSA can easily figure out what $p_1 \oplus p_2$ is! Information leaked!

OTP: Pros and Cons

Why is OTP secure?

Suppose I have the ciphertext c, but not the key or the plaintext. Can I find out anything about the plaintext?No!

For every possible plaintext p, there exists a key k such that $c = p \oplus k$. Why? Just let $k = c \oplus p$.

What's wrong with OTP?

- Need a really long key. Same length as input! Fine for credit card numbers, maybe not so fine for a few TB of top-secret blueprints for your next supervillain base...
- Can't reuse key twice without leaking info. Let's say I send $p_1 \oplus k$ and $p_2 \oplus k$. Then NSA can easily figure out what $p_1 \oplus p_2$ is! Information leaked!
- Needs a key to be shared before the transmission is done. If I need to walk into Amazon HQ to give them a secret key before sending them my CC number, why not just go to a store?

Addressing OTP Shortcomings

Long keys can be addressed with ”pseudorandom generators” that take short random strings and turn them into longer strings that ”look random”.

Addressing OTP Shortcomings

Long keys can be addressed with ”pseudorandom generators” that take short random strings and turn them into longer strings that ”look random”. Beyond the scope of this course (take CS276).

Addressing OTP Shortcomings

Long keys can be addressed with ”pseudorandom generators” that take short random strings and turn them into longer strings that ”look random”. Beyond the scope of this course (take CS276).

Address the security concerns with **public key crypto** (now).

Addressing OTP Shortcomings

Long keys can be addressed with ”pseudorandom generators” that take short random strings and turn them into longer strings that ”look random”. Beyond the scope of this course (take CS276).

Address the security concerns with **public key crypto** (now).

Big idea: Amazon gives everyone a mathematical safe that they can put stuff into, but can’t unlock.

RSA Algorithm

Formally: Amazon broadcasts a **public key** that anyone can use to encrypt data with. Amazon has (and keeps secret) a **private key** that they can use to decrypt data that's been encrypted with the public key.

RSA Algorithm

Formally: Amazon broadcasts a **public key** that anyone can use to encrypt data with. Amazon has (and keeps secret) a **private key** that they can use to decrypt data that's been encrypted with the public key.

Key generation: Amazon picks two large primes, p and q, and lets $N = pq$. It also chooses some e relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$.

RSA Algorithm

Formally: Amazon broadcasts a **public key** that anyone can use to encrypt data with. Amazon has (and keeps secret) a **private key** that they can use to decrypt data that's been encrypted with the public key.

Key generation: Amazon picks two large primes, p and q, and lets $N = pq$. It also chooses some e relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$.

Puts $N = pq$ and e on their website. Locks up d deep in the bowels of corporate HQ.

RSA Algorithm

Formally: Amazon broadcasts a **public key** that anyone can use to encrypt data with. Amazon has (and keeps secret) a **private key** that they can use to decrypt data that's been encrypted with the public key.

Key generation: Amazon picks two large primes, p and q, and lets $N = pq$. It also chooses some e relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$.

Puts $N = pq$ and e on their website. Locks up d deep in the bowels of corporate HQ.

Encrypt: Given plaintext x (say, a credit card number), David computes the ciphertext $c = E(x) = mod(x^e, N)$ and sends it to Amazon (over an open channel that NSA may be watching).

RSA Algorithm

Formally: Amazon broadcasts a **public key** that anyone can use to encrypt data with. Amazon has (and keeps secret) a **private key** that they can use to decrypt data that's been encrypted with the public key.

Key generation: Amazon picks two large primes, p and q, and lets $N = pq$. It also chooses some e relatively prime to $(p-1)(q-1)$ (normally small, say, 3), and then computes $d = e^{-1} \mod (p-1)(q-1)$.

Puts $N = pq$ and e on their website. Locks up d deep in the bowels of corporate HQ.

Encrypt: Given plaintext x (say, a credit card number), David computes the ciphertext $c = E(x) = mod(x^e, N)$ and sends it to Amazon (over an open channel that NSA may be watching).

Decrypt: Amazon computes $D(c) = mod(c^d, N)$. We'll show (next slide) this actually gives the plaintext x back.

Correctness of RSA

Theorem: For the encryption/decryption protocol on the previous slide, $D(E(x)) = x \pmod{N}$ for all $x \in \{0, 1, ...n-1\}$.

Correctness of RSA

Theorem: For the encryption/decryption protocol on the previous slide, $D(E(x)) = x \pmod{N}$ for all $x \in \{0, 1, ...n-1\}$.

Proof: It suffices to show: $(\cancel{p}\,x^e)^d \equiv \cancel{p}\,x \pmod{n}$ for all $\cancel{p}\,x \in \{0, 1, ...n-1\}$.

Correctness of RSA

Theorem: For the encryption/decryption protocol on the previous slide, $D(E(x)) = x \pmod{N}$ for all $x \in \{0, 1, \ldots n-1\}$.

Proof: It suffices to show: $(\cancel{p}^e)^d = \cancel{p} \pmod{n}$ for all $\cancel{p} \in \{0, 1, \ldots n-1\}$.

Consider the exponent ed. We kow that $ed \equiv 1 \mod (p-1)(q-1)$ by definition, so $ed = 1 + k(p-1)(q-1)$ for some integer k.

Correctness of RSA

Theorem: For the encryption/decryption protocol on the previous slide, $D(E(x)) = x \pmod N$ for all $x \in \{0, 1, ...n-1\}$.

Proof: It suffices to show: $(\cancel{p}\,x^e)^d = \cancel{p}\,x \pmod n$ for all $\cancel{p}\,x \in \{0, 1, ...n-1\}$.

Consider the exponent ed. We kow that $ed \equiv 1 \mod (p-1)(q-1)$ by definition, so $ed = 1 + k(p-1)(q-1)$ for some integer k. Therefore,

$$x^{ed} - x = x^{1+k(p-1)(q-1)} - x = x(x^{k(p-1)(q-1)} - 1) \quad .$$

Correctness of RSA

Theorem: For the encryption/decryption protocol on the previous slide, $D(E(x)) = x \pmod{N}$ for all $x \in \{0, 1, \ldots n-1\}$.

Proof: It suffices to show: $(\cancel{p}\,x^e)^d = \cancel{p}\,x \pmod{n}$ for all $\cancel{p}\,x \in \{0, 1, \ldots n-1\}$.

Consider the exponent ed. We kow that $ed \equiv 1 \mod (p-1)(q-1)$ by definition, so $ed = 1 + k(p-1)(q-1)$ for some integer k. Therefore,

$$x^{ed} - x = x^{1+k(p-1)(q-1)} - x = x(x^{k(p-1)(q-1)} - 1) \quad .$$

Suffices to show that this expression is 0 mod N for all x, i.e. that it's a multiple of both p and q. We will show it's a multiple of p.

Theorem: For the encryption/decryption protocol on the previous slide, $D(E(x)) = x \pmod N$ for all $x \in \{0, 1, ...n-1\}$.

Proof: It suffices to show: $(\cancel{p}^e)^d = \cancel{p} \pmod n$ for all $\cancel{p} \in \{0, 1, ...n-1\}$.

Consider the exponent ed. We kow that $ed \equiv 1 \mod (p-1)(q-1)$ by definition, so $ed = 1 + k(p-1)(q-1)$ for some integer k. Therefore,

$$x^{ed} - x = x^{1+k(p-1)(q-1)} - x = x(x^{k(p-1)(q-1)} - 1) \quad .$$

Suffices to show that this expression is 0 mod N for all x, i.e. that it's a multiple of both p and q. We will show it's a multiple of p.

$a^{bc} = (a^b)^c$

- Case 1: p divides x. Then obviously it also divides $x(x^{k(p-1)(q-1)} - 1)$, as desired.
- Case 2: p doesn't divide x. Then $x^{k(p-1)(q-1)} = (x^{p-1})^{k(q-1)}$. Applying Fermat's little theorem, $x^{p-1} \equiv 1 \pmod p$. So $x^{k(p-1)(q-1)} - 1 \equiv 1^{k(q-1)} - 1 \equiv 0 \pmod p$, so $x(x^{k(p-1)(q-1)} - 1)$ must be a multiple of p.

Argument for q is exactly the same. Therefore $q\,p|(x^{ed} - x)$. $\square$

On the Security of RSA

Why is RSA secure? Even without the private key, we have enough information to decrypt anything we see (we could just take the public key, encrypt every possible string representable as a number under N, and see which one matches the ciphertext).

Why is RSA secure? Even without the private key, we have enough information to decrypt anything we see (we could just take the public key, encrypt every possible string representable as a number under N, and see which one matches the ciphertext).

The security RSA, like all almost all encryption schemes, relies on *hardness assumptions*. We need to assume something is hard in order to show that decrypting something, or even getting some information about the plaintext, *even with full information*, is hard.

Message Indistinguishability*

How do we formalize this notion of ”hard to get information about the plaintext”?

Quasi-formally: under some hardness assumptions, this must hold for *all* pairs of strings $m^{(1)}$, $m^{(0)}$: for any *probabilistically polynomial time* (”PPT”) algorithm A that knows the length of the strings and the public key, the probability that A returns 1 given the public key and the encryption of $m^{(1)}$ must be ”extremely close” to the probability that it returns 1 on encryption of $m^{(0)}$.

Message Indistinguishability*

How do we formalize this notion of ”hard to get information about the plaintext”?

Quasi-formally: under some hardness assumptions, this must hold for *all* pairs of strings $m^{(1)}$, $m^{(0)}$: for any *probabilistically polynomial time* (”PPT”) algorithm A that knows the length of the strings and the public key, the probability that A returns 1 given the public key and the encryption of $m^{(1)}$ must be ”extremely close” to the probability that it returns 1 on $m^{(0)}$. Formally:

$$\left|\Pr[A^{E(1^k,PK)}(1^k, PK, E(1^k, PK, m_k^{(1)})) = 1] - \Pr[A^{E(1^k,PK)}(1^k, PK, E(1^k, PK, m_k^{(0)})) = 1]\right|$$

is “negligible” in k.

Intuitively? There is no algorithm (even if we allow the algorithm access to the public key) that runs in a reasonable amount of time that can distinguish between the ciphertexts for two different plaintexts.

Message Indistinguishability*

How do we formalize this notion of ”hard to get information about the plaintext”?

Quasi-formally: under some hardness assumptions, this must hold for *all* pairs of strings $m^{(1)}$, $m^{(0)}$: for any *probabilistically polynomial time* (”PPT”) algorithm A that knows the length of the strings and the public key, the probability that A returns 1 given the public key and the encryption of $m^{(1)}$ must be ”extremely close” to the probability that it returns 1 on $m^{(0)}$. Formally:

$$\left|\Pr[A^{E(1^k,PK)}(1^k, PK, E(1^k, PK, m_k^{(1)}) = 1] - \Pr[A^{E(1^k,PK)}(1^k, PK, E(1^k, PK, m_k^{(0)}) = 1]\right|$$

is “negligible” in k.

Intuitively? There is no algorithm (even if we allow the algorithm access to the public key) that runs in a reasonable amount of time that can distinguish between the ciphertexts for two different plaintexts. “Message indistinguishability under chosen plaintext attack”.

Hardness Assumptions

What hardness assumptions are we making for RSA?

Hardness Assumptions

What hardness assumptions are we making for RSA?

"Given N, e, $c = x^e \pmod{N}$, there is ~~an~~ no efficient algorithm for determining x."

Hardness Assumptions

What hardness assumptions are we making for RSA?

"Given N, e, $c = x^e \pmod{N}$, there is on efficient algorithm for determining x."

How would the NSA guess x?

- Brute force: try encrypting every possible string x. There are too many values of x - $2^{|x|}$. Can't do this efficiently.

Hardness Assumptions

What hardness assumptions are we making for RSA?

"Given N, e, $c = x^e \pmod{N}$, there is on efficient algorithm for determining x."

How would the NSA guess x?

- Brute force: try encrypting every possible string x. There are too many values of x - $2^{|x|}$. Can't do this efficiently.
- Factoring: Try determining d by factoring N into p and q, which would allow NSA to compute d the same way Amazon did. Factoring large numbers is considered impossible to do efficiently.

Hardness Assumptions

What hardness assumptions are we making for RSA?

"Given N, e, $c = x^e \pmod{N}$, there is on efficient algorithm for determining x."

How would the NSA guess x?

- Brute force: try encrypting every possible string x. There are too many values of x - $2^{|x|}$. Can't do this efficiently.
- Factoring: Try determining d by factoring N into p and q, which would allow NSA to compute d the same way Amazon did. Factoring large numbers is considered impossible to do efficiently.
- Direct computation of $(p-1)(q-1)$. Reduces to factoring. Why? If you compute $(p-1)(q-1) = pq - p - q + 1$, you now know what $p+q$ and pq are. Trivial to solve for p and q from here.

Hardness Assumptions

What hardness assumptions are we making for RSA?

"Given N, e, $c = x^e \pmod{N}$, there is on efficient algorithm for determining x."

How would the NSA guess x?

- Brute force: try encrypting every possible string x. There are too many values of x - $2^{|x|}$. Can't do this efficiently.
- Factoring: Try determining d by factoring N into p and q, which would allow NSA to compute d the same way Amazon did. Factoring large numbers is considered impossible to do efficiently.
- Direct computation of $(p-1)(q-1)$. Reduces to factoring. Why? If you compute $(p-1)(q-1) = pq - p - q + 1$, you now know what $p+q$ and pq are. Trivial to solve for p and q from here.

Security of breaking RSA requires on hardness of factoring large integers.

Prime-Finding

RSA also relies on the ability to find large primes p and q. How do we do that?

Prime-Finding

RSA also relies on the ability to find large primes p and q. How do we do that?

Prime number theorem: Let $\pi(x)$ denote the number of prime numbers less than or equal to x. Then as x goes to infinity, $\pi(x)$ converges to $x/\ln x$.

Proof: Many of them, but all of them require math far beyond the scope of this course.

Prime-Finding

RSA also relies on the ability to find large primes p and q. How do we do that?

Prime number theorem: Let $\pi(x)$ denote the number of prime numbers less than or equal to x. Then as x goes to infinity, $\pi(x)$ converges to $x/\ln x$.

Proof: Many of them, but all of them require math far beyond the scope of this course.

Main takeaway: primes aren't too uncommon. If we select a few hundred 512-bit numbers, there will probably be a prime among them.

Prime-Finding

RSA also relies on the ability to find large primes p and q. How do we do that?

Prime number theorem: Let $\pi(x)$ denote the number of prime numbers less than or equal to x. Then as x goes to infinity, $\pi(x)$ converges to $x/\ln x$.

Proof: Many of them, but all of them require math far beyond the scope of this course.

Main takeaway: primes aren't too uncommon. If we select a few hundred 512-bit numbers, there will probably be a prime among them.

Problem: how do we figure out if something's a prime?

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$?

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$? Then k can't be prime!

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$? Then k can't be prime! $a^{k-1} \not\equiv 1 \pmod{k}$

Suppose k is composite. Call a such that $a^{k-1} \not\equiv 1 \pmod{k}$ "Fermat witnesses" and a such that $a^{k-1} \equiv 1 \pmod{k}$ "Fermat liars".

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$? Then k can't be prime!
$a^{k-1} \not\equiv 1 \pmod{k}$

Suppose k is composite. Call a such that $a^{k-1} \not\equiv 1 \pmod{k}$ "Fermat witnesses" and a such that $a^{k-1} \equiv 1 \pmod{k}$ "Fermat liars". Suppose we have one Fermat witness. There must be at least one Fermat witness for each Fermat liar. Why?

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$? Then k can't be prime! $a^{k-1} \not\equiv 1 \pmod{k}$

Suppose k is composite. Call a such that $a^{k-1} \not\equiv 1 \pmod{k}$ "Fermat witnesses" and a such that $a^{k-1} \equiv 1 \pmod{k}$ "Fermat liars". Suppose we have one Fermat witness. There must be at least one Fermat witness for each Fermat liar. Why?

Let's say a is a Fermat witness and $b_1, \ldots, b_l$ are a Fermat liar. Then

$$(ab_i)^{k-1} \equiv a^{k-1} b_i^{k-1} \equiv a^{k-1} 1 \not\equiv 1 \pmod{k} .$$

So we have a list of l Fermat witnesses.

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$? Then k can't be prime! $a^{k-1} \not\equiv 1 \pmod{k}$

Suppose k is composite. Call a such that $a^{k-1} \not\equiv 1 \pmod{k}$ "Fermat witnesses" and a such that $a^{k-1} \equiv 1 \pmod{k}$ "Fermat liars". Suppose we have one Fermat witness. There must be at least one Fermat witness for each Fermat liar. Why?

Let's say a is a Fermat witness and $b_1, ..., b_l$ are a Fermat liar. Then

$$(ab_i)^{k-1} \equiv a^{k-1} b_i^{k-1} \equiv a^{k-1} 1 \not\equiv 1 \pmod{k} \ .$$

So we have a list of l Fermat witnesses.

If we pick random a and k is composite: probability that we say "prime" is $a^{k-1} \not\equiv 1 \pmod{k}$ is at least 1/2.

A Simple Primality Test

Recall Fermat's little theorem: if p is prime and $1 \leq a \leq p$, then $a^{p-1} \equiv 1 \pmod{p}$.

What if we see that $a^{k-1} \not\equiv 1 \pmod{k}$? Then k can't be prime! $a^{k-1} \not\equiv 1 \pmod{k}$

Suppose k is composite. Call a such that $a^{k-1} \not\equiv 1 \pmod{k}$ "Fermat witnesses" and a such that $a^{k-1} \equiv 1 \pmod{k}$ "Fermat liars". Suppose we have one Fermat witness. There must be at least one Fermat witness for each Fermat liar. Why?

Let's say a is a Fermat witness and $b_1, \ldots, b_l$ are a Fermat liar. Then

$$(ab_i)^{k-1} \equiv a^{k-1}b_i^{k-1} \equiv a^{k-1}1 \not\equiv 1 \pmod{k} \quad .$$

So we have a list of l Fermat witnesses.

If we pick random a and k is composite: probability that we say "prime" is $a^{k-1} \not\equiv 1 \pmod{k}$ is at least $1/2$. Pick n random numbers to reduce false prime reporting rate to $1/2^n$.

Carmichael Numbers

What if we can't assume that there is a Fermat ?

Carmichael Numbers

What if we can't assume that there is a Fermat witness? Carmichael numbers! Composites where *all* a for which $\gcd(a, k) = 1$ are Fermat liars.

Carmichael numbers are a good deal rarer than primes but can still be a problem.

What if we can't assume that there is a Fermat ? Carmichael numbers! Composites where *all a* for which $\gcd(a, k) = 1$ are Fermat liars.

Carmichael numbers are a good deal rarer than primes but can still be a problem. There are better primality tests that extend Fermat's to deal with Carmichael numbers: Miller-Rabin, Bailie-PSW, Solovay-Strassen.

Carmichael Numbers

What if we can't assume that there is a Fermat ? Carmichael numbers! Composites where *all a* for which $\gcd(a, k) = 1$ are Fermat liars.

Carmichael numbers are a good deal rarer than primes but can still be a problem. There are better primality tests that extend Fermat's to deal with Carmichael numbers: Miller-Rabin, Bailie-PSW, Solovay-Strassen. Often Fermat's primality test is used to filter out obvious non-primes before one of these other (slower) tests is used.

Aside: Derandomization and Complexity*

Can you find big primes without randomness? Yes!

Aside: Derandomization and Complexity*

Can you find big primes without randomness? Yes!

AKS primality test [Agrawal–Kayal–Saxena '02]: you can find primes "efficiently" (roughly # of bits to the sixth power) without using randomness.

Aside: Derandomization and Complexity*

Can you find big primes without randomness? Yes!

AKS primality test [Agrawal–Kayal–Saxena '02]: you can find primes "efficiently" (roughly # of bits to the sixth power) without using randomness.

Fundamental question in computer science: how much additional computational power does randomness give you? Can you do things with randomness efficiently that you can't without randomness?

Aside: Derandomization and Complexity*

Can you find big primes without randomness? Yes!

AKS primality test [Agrawal–Kayal–Saxena '02]: you can find primes "efficiently" (roughly # of bits to the sixth power) without using randomness.

Fundamental question in computer science: how much additional computational power does randomness give you? Can you do things with randomness efficiently that you can't without randomness?

Major open problem! There are problems that we know how to solve with randomness, but we don't know how to solve deterministically.

The Chinese Remainder Theorem, Euler's Criterion, and an Application to Flipping Coins

Simultaneous Congruences

Theorem: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod mn.

Simultaneous Congruences

Theorem: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod mn.

Proof: To satisfy the first equation: we must have $x = a + mt$ for some integer t.

Simultaneous Congruences

Theorem: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod mn.

Proof: To satisfy the first equation: we must have $x = a + mt$ for some integer t.

To satisfy the second equation we must have $x \equiv a + mt \equiv b \pmod{n}$, or $mt \equiv b - a \pmod{n}$.

Simultaneous Congruences

Theorem: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod mn.

Proof: To satisfy the first equation: we must have $x = a + mt$ for some integer t.

To satisfy the second equation we must have $x \equiv a + mt \equiv b \pmod{n}$, or $mt \equiv b - a \pmod{n}$. $t = m^{-1}(b-a)$

Since $\gcd(m, n) = 1$, m has a multiplicative inverse mod n, so we can determine t uniquely mod n. Let's say $t \equiv c \pmod{n}$. So there exists integer k such that $t = c + nk$.

Simultaneous Congruences

Theorem: Suppose $\gcd(m, n) = 1$. Then the two equations $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ have a unique solution mod mn.

Proof: To satisfy the first equation: we must have $x = a + mt$ for some integer t.

To satisfy the second equation we must have $x \equiv a + mt \equiv b \pmod{n}$, or $mt \equiv b - a \pmod{n}$.

Since $\gcd(m, n) = 1$, m has a multiplicative inverse mod n, so we can determine t uniquely mod n. Let's say $t \equiv c \pmod{n}$. So there exists integer k such that $t = c + nk$.

So $x = a + m(c + nk) = (a + mc) + mnk$, i.e. $x \equiv \overset{a+mc}{\cancel{m(c + nk)}} \pmod{mn}$; this is a unique solution to the equations mod mn. $\square$

Chinese Remainder Theorem

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, \ldots, m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}$, ..., $x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 \ldots m_k$.

Chinese Remainder Theorem

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, ..., m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}$, ..., $x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 ... m_k$.

Proof:

Chinese Remainder Theorem

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, \ldots, m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}$, ..., $x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 \ldots m_k$.

Proof: by induction on k.

Chinese Remainder Theorem

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, ..., m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}$, ..., $x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 ... m_k$.

Proof: by induction on k.

For the base case, let $k = 2$. This is just the theorem on the previous page.

Chinese Remainder Theorem

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, \ldots, m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}, \ldots, x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 \ldots m_k$.

Proof: by induction on k.

For the base case, let $k = 2$. This is just the theorem on the previous page.

Now suppose for induction that the theorem holds for up to k equations. We wish to show that it holds for $k + 1$ equations.

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, ..., m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}$, ..., $x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 ... m_k$.

Proof: by induction on k.

For the base case, let $k = 2$. This is just the theorem on the previous page.

Now suppose for induction that the theorem holds for up to k equations. We wish to show that it holds for $k + 1$ equavions.

Remove the $k + 1$st equation. We have k equations, which (by inductive hypothesis) have a unique solution mod $m_1 m_2 ... m_k$, i.e. $x = t \pmod{m_1 m_2 ... m_k}$.

Chinese Remainder Theorem

We can generalize this to multiple primes!

Chinese Remainder Theorem: Let $m_1, \ldots, m_k$ be relatively prime numbers. Then the k equations $x \equiv a_1 \pmod{m_1}, \ldots, x \equiv a_k \pmod{m_k}$ have a unique solution mod $m_1 m_2 \ldots m_k$.

Proof: by induction on k.

For the base case, let $k = 2$. This is just the theorem on the previous page.

Now suppose for induction that the theorem holds for up to k equations. We wish to show that it holds for $k + 1$ equavions.

Remove the $k + 1$st equation. We have k equations, which (by inductive hypothesis) have a unique solution mod $m_1 m_2 \ldots m_k$, i.e. $x \equiv t \pmod{m_1 m_2 \ldots m_k}$.

Add the last equation back. Since m_{k+1} is relatively prime to each of $m_1, \ldots, m_k$, it is relatively prime to $m_1 m_2 \ldots m_k$. So by the previous theorem, there is a unique solution mod $(m_1 m_2 \ldots m_k) m_{k+1}$. $\square$

Euler’s Criterion

BLANK

SLIDE

Flipping Coins Remotely

Suppose Alex and David want flip a coin, but they're a country apart. Alex bets on heads and David bets on tails. How do they flip a coin fairly?

Flipping Coins Remotely

Suppose Alex and David want flip a coin, but they're a country apart. Alex bets on heads and David bets on tails. How do they flip a coin fairly?

Problem: suppose neither side trusts the other to be honest.

David: "I flipped a coin and got tails." Alex: "You're just saying that because you want tails."

Flipping Coins Remotely

Suppose Alex and David want flip a coin, but they're a country apart. Alex bets on heads and David bets on tails. How do they flip a coin fairly?

Problem: suppose neither side trusts the other to be honest.

David: "I flipped a coin and got tails." Alex: "You're just saying that because you want tails."

How do you do this in a way that doesn't require trust on both sides?

Flipping Coins Remotely

Suppose Alex and David want flip a coin, but they're a country apart. Alex bets on heads and David bets on tails. How do they flip a coin fairly?

Problem: suppose neither side trusts the other to be honest.

David: "I flipped a coin and got tails." Alex: "You're just saying that because you want tails."

How do you do this in a way that doesn't require trust on both sides?

Number theory to the rescue!

Square Roots in Modular Arithmetic

Theorem (Euler's Criterion): Suppose p is an odd prime and a is some integer relatively prime to p. Then $a^{(p-1)/2}$ is 1 mod p if and only if there exists some integer x such that $a \equiv x^2 \pmod{p}$ and -1 otherwise.

Square Roots in Modular Arithmetic

Theorem (Euler's Criterion): Suppose p is an odd prime and a is some integer relatively prime to p. Then $a^{(p-1)/2}$ is 1 if and only if there exists some integer x such that $a \equiv x^2 \pmod{p}$ and -1 otherwise.

Proof: If direction:

$$a^{(p-1)/2} = (x^2)^{(p-1)/2} = x^{p-1} \equiv 1 \pmod{p}$$

by Fermat's little theorem.

Only if direction: more complicated, but we won't use (or prove) it here. □

Square Roots in Modular Arithmetic

Theorem (Euler's Criterion): Suppose p is an odd prime and a is some integer relatively prime to p. Then $a^{(p-1)/2}$ is 1 if and only if there exists some integer x such that $a \equiv x^2 \pmod{p}$ and -1 otherwise.

Proof: If direction:

$$a^{(p-1)/2} = (x^2)^{(p-1)/2} = x^{p-1} \equiv 1 \pmod{p}$$

by Fermat's little theorem.

Only if direction: more complicated, but we won't use (or prove) it here. $\square$

Notice that if $a \equiv 3 \pmod{4}$, then we can find square roots easily.

Square Roots in Modular Arithmetic

Theorem (Euler's Criterion): Suppose p is an odd prime and a is some integer relatively prime to p. Then $a^{(p-1)/2}$ is 1 if and only if there exists some integer x such that $a \equiv x^2 \pmod{p}$ and -1 otherwise.

Proof: If direction:

$$a^{(p-1)/2} = (x^2)^{(p-1)/2} = x^{p-1} \equiv 1 \pmod{p}$$

by Fermat's little theorem.

Only if direction: more complicated, but we won't use (or prove) it here. □

Notice that if $a \equiv 3 \pmod{4}$, then we can find square roots easily. In fact, if the solutions to $x^2 \equiv a \pmod{p}$ are given by $x \equiv \pm a^{(p+1)/4} \pmod{p}$. Why?

$$(\pm a^{(p+1)/4})^2 \equiv a^{(p+1)/2} \equiv a^{(p-1)/2} a \equiv 1a \equiv a \pmod{p}$$

.

Square roots mod pq

Suppose $x^2 \equiv a \pmod{pq}$. Then we must have $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$.

Square roots mod pq

Suppose $x^2 \equiv a \pmod{pq}$. Then we must have $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$.

The first congruence gives us $x \equiv \pm x_1 \pmod{p}$; the second gives us $x \equiv \pm x_2 \pmod{q}$.

Square roots mod pq

Suppose $x^2 \equiv a \pmod{pq}$. Then we must have $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$.

The first congruence gives us $x \equiv \pm x_1 \pmod{p}$; the second gives us $x \equiv \pm x_2 \pmod{q}$.

Four sets of equations (choose a sign for the p, and the q.)

Square roots mod pq

Suppose $x^2 \equiv a \pmod{pq}$. Then we must have $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$.

The first congruence gives us $x \equiv \pm x_1 \pmod{p}$; the second gives us $x \equiv \pm x_2 \pmod{q}$.

Four sets of equations (choose a sign for the p, and the q.)

One unique solution to each set of equations by the Chinese remainder theorem.

Square roots mod pq

Suppose $x^2 \equiv a \pmod{pq}$. Then we must have $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$.

The first congruence gives us $x \equiv \pm x_1 \pmod{p}$; the second gives us $x \equiv \pm x_2 \pmod{q}$.

Four sets of equations (choose a sign for the p, and the q.)

One unique solution to each set of equations by the Chinese remainder theorem.

Four square roots mod pq!

Square roots mod pq

Suppose $x^2 \equiv a \pmod{pq}$. Then we must have $x^2 \equiv a \pmod{p}$ and $x^2 \equiv a \pmod{q}$.

The first congruence gives us $x \equiv \pm x_1 \pmod{p}$; the second gives us $x \equiv \pm x_2 \pmod{q}$.

Four sets of equations (choose a sign for the p, and the q.)

One unique solution to each set of equations by the Chinese remainder theorem.

Four square roots mod pq!

Combine sqare root formula on previous slide for single prime congruent to $3 \pmod{4}$ with trick here gives us an easy way to compute square roots of numbers mod pq where p, q are congruent to $3 \pmod{4}$.

Products of distinct primes both congruent to $3 \pmod{4}$ are called "Blum integers".

Blum's Coin-Flipping Scheme

Here's how to flip a coin over the telephone [Blum-'82]:

1. Alex chooses distinct primes p, q congruent to $3 \pmod 4$, and computes $n = pq$. He sends n (but not p and q) to David.

Blum's Coin-Flipping Scheme

Here's how to flip a coin over the telephone [Blum-'82]:

1. Alex chooses distinct primes p, q congruent to $3 \pmod 4$, and computes $n = pq$. He sends n (but not p and q) to David.
2. David chooses $x \in (0, n)$ relatively prime to n and sends $a = x^2 \pmod n$ to Alex.

Blum's Coin-Flipping Scheme

Here's how to flip a coin over the telephone [Blum-'82]:

1. Alex chooses distinct primes p, q congruent to $3 \pmod 4$, and computes $n = pq$. He sends n (but not p and q) to David.
2. David chooses $x \in (0, n)$ relatively prime to n and sends $a = x^2 \pmod n$ to Alex.
3. Alex, armed with knowledge of p, q, computes the square roots $\pm x, \pm y$ of a, mod n, and sends one to David.

Blum's Coin-Flipping Scheme

Here's how to flip a coin over the telephone [Blum-'82]:

1. Alex chooses distinct primes p, q congruent to $3 \pmod 4$, and computes $n = pq$. He sends n (but not p and q) to David.
2. David chooses $x \in (0, n)$ relatively prime to n and sends $a = x^2 \pmod n$ to Alex.
3. Alex, armed with knowledge of p, q, computes the square roots $\pm x, \pm y$ of a, mod n, and sends one to David.
4. If David got $\pm x$, then he says Alex ~~Bob~~ guessed correctly. Otherwise, if he gets $\pm y$, he can factor n and use that to prove that he won.

Blum’s Coin-Flipping Scheme: Analysis

Alex has no idea whether David chose x or y, so he has a 1/2 chance of picking x.

Blum's Coin-Flipping Scheme: Analysis

Alex has no idea whether David chose x or y, so he has a 1/2 chance of picking x.

If David got $\pm y$: he now has two different square roots of $a \pmod{n}$. Now he can use this to factor n: since $x^2 \equiv a \equiv y^2 \pmod{n}$ (with x, y distinct), $pq | (x+y)(x-y)$, so each prime divides either $(x+y)$ or $(x-y)$ but not both. So $\gcd(x+y, n)$ and $\gcd(x-y, n)$ provide the two prime factors. All David has to do is compute $x^2 - y^2 = (x+y)(x-y)$ and run EGCD twice!

Blum's Coin-Flipping Scheme: Analysis

Alex has no idea whether David chose x or y, so he has a 1/2 chance of picking x.

If David got $\pm y$: he now has two different square roots of $a \pmod{n}$. Now he can use this to factor n: since $x^2 \equiv a \equiv y^2 \pmod{n}$ (with x, y distinct), $pq | (x+y)(x-y)$, so each prime divides either $(x+y)$ or $(x-y)$ but not both. So $\gcd(x+y, n)$ and $\gcd(x-y, n)$ provide the two prime factors. All David has to do is compute $x^2 - y^2 = (x+y)(x-y)$ and run EGCD twice!

If David got $\pm x$: he's learned nothing, so he can't factor ~~p~~ n any better than brute force (which is hard).

Blum's Coin-Flipping Scheme: Analysis

Alex has no idea whether David chose x or y, so he has a 1/2 chance of picking x.

If David got $\pm y$: he now has two different square roots of $a \pmod{n}$. Now he can use this to factor n: since $x^2 \equiv a \equiv y^2 \pmod{n}$ (with x, y distinct), $pq|(x+y)(x-y)$, so each prime divides either $(x+y)$ or $(x-y)$ but not both. So $\gcd(x+y, n)$ and $\gcd(x-y, n)$ provide the two prime factors. All David has to do is compute $x^2 - y^2 = (x+y)(x-y)$ and run EGCD twice!

If David got $\pm x$: he's learned nothing, so he can't factor p any better than brute force (which is hard).

After the game is over each side can verify the other's honesty: David asks Alex for the factors p, q to make sure they're Blum integers and check that they're primes.

Questions?