Randomness and Cryptography

Yevgeniy Dodis

New York University

Popular Children's Game

- Each child chooses scissors, paper or rock
  - Rock beats Scissors
  - Scissors beat Paper
  - Paper beats Rock
- What do we play?
  - For any strategy there is a better response !
- Solution: play AT RANDOM
  - No matter what the opponent does, break even on average !
- Randomization essential here

Randomness

- Crucial in many areas:
  - Approximation algorithms
  - Distributed computing
  - Property testing
  - Counting problems
  - Symmetry breaking
  - Reducing Complexity (sampling, embedding)
  - Game theory
  - Cryptography
  - …

The Big Picture

1. Reasons for Randomness in Crypto
2. Imperfect Sources of Randomness
3. Cryptography from entropy alone?
   - Does crypto requires extraction?
4. Randomness Extraction
   - Variants: fuzzy, local, interactive,…
5. Leakage-Resilient Cryptography
6. Pseudorandomness

The Big Picture

1. Reasons for Randomness in Crypto
2. Imperfect Sources of Randomness
3. Cryptography from entropy alone?
   - Does crypto requires extraction?
4. Randomness Extraction
   - Variants: fuzzy, local, interactive,…
5. Leakage-Resilient Cryptography
6. Pseudorandomness

Randomness in Crypto

- Unlike many other examples, randomness essential for security!
- Secret keys have to be random
  - If not, everything is easy
- Security against "replay" attacks (e.g., challenge-response, encryption, …)
- Privacy and Anonymity
  - Many examples (stay tuned)
- Unpredictability (e.g., of challenges, fingerprints, …)

Key generation 1

- Toy example: Ceasar cipher
- Enc(letter) = letter + 3 mod 26
  - Enc(RANDOM) = UDQGRP
  - Can't rely on secrecy of algorithms, only of secret keys (Keirkhoff's principle)!
- Example 2: shift cipher
- $Enc_s(x) = x+s \bmod 26$
  - Key is too short, only 26 keys !
  - Keys must have enough entropy to defeat brute force attacks !

Key generation 2

- Example 3: permutation cipher
- Enc_π(letter) = π(letter), where π is random permutation
  - Enc_π(NOT GOOD) = RZP BZZQ
  - # keys = 26! = 2^{95} (large enough)
  - Not good, see that same letter "Z" appeared three times (frequency analysis kills it)
  - Entropy alone is not enough ! (more later)
  - Need to have precise goal and argue that your system meets it

Key generation 3

- Example 3: one-time pad
- <u>Goal</u>: encrypt n-bit message once and have ciphertext reveal "no information" about the plaintext (e.g., $H(M) = H(M|C)$, for any distribution on M)
  - Even the goal is probabilistic in nature !
- $Enc_K(m) = m \oplus K$, $Dec_K(c) = c \oplus K$
  - Satisfies defn, provided K is truly random (aside: $|K|=|M|$, bad but best possible ☹)
  - How crucial is this assumption?

Randomness of keys?

- If Eve knows some info about K $\Rightarrow$ translates to the same info about M !
  - E.g., $M_1 = C_1 \oplus K_1$
  - In general, partial info reduces brute force search and most cryptanalysis techniques !
- Important: assume can generate keys according to the "distribution we need" (which is typically uniformly random in the symmetric key setting)
  - Revisit this later, but assume for now !

Randomness of keys?

- What about "practical" ciphers (DES, AES, RC2, …)?
  - Often believe "any key is good"
- Dangerous
  - Ex: 0^{56}, $0^{28}1^{28}$, 1^{56}, $1^{28}0^{28}$ weak for DES
  - Not the design criteria of creators
  - Meaningless formally: any "specific" key is weak since "know" the secret key
  - Need random experiment to even make sense of security !

Randomness of keys?

- Heuristics: if K has enough entropy, practical systems based on DES, AES, … are secure.
  - No formal justification !
  - In fact, I will later give strong evidence that this is very suspect ! [DOPS04]
- Punchline: current symmetric key systems crucially rely on the randomness of their secret keys !

What about Public Keys?

- Example: ElGamal encryption.
- All we need to know is that need common prime p where discrete log (from p, g and $y = g^x \bmod p$ compute x) is "hard"
- "Great" choice: use p of the form $2^k + 1$
  - Recall, order of multiplicative group (p-1) ($z^{p-1} = 1$ for all z by Fermat's little theorem)
  - Very fast operations !
- Insecure: discrete log is easy !

Attack

- Easy to see: x_k is even iff $y^{2^{k-1}} \bmod p = 1$
- More generally x ends with j zeros iff

$$y^{2^{k-j}} \bmod p = 1$$

- For j = 0 to k-1
  - Set $x_{k-j} = 0$ iff $y^{2^{k-j-1}} \bmod p = 1$
  - If $x_{k-j} = 1$, change $y := y/g \bmod p$
- Output $x_1 \ldots x_k$

Key Generation

- Moral: every crypto system (PK or SK) has a well defined hardness assumption (or security proof) involving, among other things, generation of public/secret keys
  - Security might crucially rely on key generation performed exactly as specified by the assumption
  - Most often need uniform random data
- Discrete Log: if p is random k-bit prime, g – random generator of Z_p and x – random exponent in {1…p-1}, then hard to compute x given (p, g, $y = g^x \bmod p$)

Lessons

- Crypto depends on randomness of keys
  - Even security goal are probabilistic
- Need high-entropy, but not enough
- Most current systems need uniformly random bits (or something derived from them)
- Security might break if the key distribution is not what you expect (more later)
- Key **assumption**: assume have a source of truly random bits – will revisit later
  - separates use of randomness from generation
- **What can we do with it???**

Reasons for Randomness

- Key Generation ✓

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing

Ex. 1: Adding 2 Numbers

- Alice has a, Bob has b
- Chris needs to compute $S = a \oplus b$
- Alice does not want Chris or Bob to learn a
- Bob does not want Chris or Alice to learn b
- Alice: pick random r and send it to Bob
- Alice: Compute $a' = a \oplus r$ and send it to Chris
- Bob: Compute $b' = b \oplus r$ and send it to Chris
- Chris: compute $a' \oplus b' = a \oplus b \oplus (r \oplus r) = a \oplus b$
  - Does not give Chris any info about a, b beside sum
- Alice and Bob also only know random r

Ex. 2: Blind RSA Signature

- Recall RSA: $n = pq$, where p,q - primes
- $\varphi(n) = (p-1)(q-1)$. For any z, $z^{\varphi(n)} = 1 \bmod n$
- Pick random e and let $d = e^{-1} \bmod \varphi(n)$
- $PK = (n,e)$. $SK = d$.
- $Sig_{SK}(m) = H(m)^d \bmod n$,
  - here H is "good" hash function (not important)
- $Ver_{PK}(\sigma,m)$: Check $\sigma^e = H(m) \bmod n$
  - Indeed, $\sigma^e = H(m)^{ed} = H(m)^1 = H(m) \bmod n$
- Assume Bob knows d, Alice knows m, and wants to compute $Sig(m)$ without telling m to Bob?
  - "blind" signature, useful for e-cahs, etc. (stay tuned)

Ex. 2: Blind RSA Signature

- Alice (knows n,e,m, not d):
  - Pick random r and compute $A = r^e H(m) \bmod n$
  - Send A to Bob for signing
  - Note, A is random and independent from $H(m)$
- Bob (knows d, τ but not r, m):
  - Compute $\tau = A^d \bmod n$ and send τ to Alice
  - Note, $\tau = A^d = r^{ed} H(m)^d = r\, H(m)^d = r\sigma \bmod n$
- Alice:
  - Compute $\sigma = \tau r^{-1} = H(m)^d \bmod n$

Randomness for Privacy

- We will see many other examples: encryption, commitment, zero-knowledge,…
- Perhaps most important use in crypto
- Strongly requires uniform randomness
  - i.e., non-uniform pads, masks, etc. leak partial information
- We will see later that it is very hard (impossible?) to securely realize such applications without perfect randomness

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)

Unpredictability

- Do not want the attacker to guess some information before it becomes available
- Example: identification
- Alice wants to prove she is "Alice" to Bob
- Naturally, Bob should "challenge" Alice with stuff only Alice should know
- If predictable challenges
  - Eve might know what Bob expects
  - Perhaps Eve convinces Alice to identify herself before she would do it to Bob. Then can simply forward Alice's answer later.

Doing It With Encryption

- Alice has (SK, PK) for an encryption scheme
- Bob: chooses random R, lets $c = Enc_{PK}(R)$ and challenges Alice with c
- Alice: computes $R' = Dec_{SK}(c)$ and sends R' to Bob
- Bob: accepts if R = R'
- Intuition: only Alice can decrypt
- Clearly, insecure if Eve can predict R
  - just send R to Bob ignoring c !
- Conversely, can show "secure" if (1) Enc is "strong enough" and (2) R is unpredictable:
  - for any $R_1 \ldots R_k$, $Pr_R(\exists i\ R = R_i) = $ tiny

Doing It With Signatures

- Alice has (SK,PK) for a signature scheme.
- Bob: send random R to Alice
- Alice: returns $Sig_{SK}(R)$
- Bob: accepts if correct $Ver_{PK}(\sigma, R)$ = true
- Intuition: only Alice can sign
- Assume Eve convinces Alice to identify herself before she would do it to Bob
  - If R is predicatable, Eve can send same R to Alice and learn Sig(R) !
- Conversely, unpredictability (+ good signature) are enough for security

More on Unpredictability

- Does not require true randomness!
  - High entropy is necessary and sufficient !
- As we will see, this makes this use of randomness more realistic than requiring perfect randomness
- Look-ahead questions:
  - can we get perfect randomness from high entropy one? (mixed answer, mainly NO)
  - What about computational unpredictability, where it is only "hard to predict"?

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)
- Freshness (non-repeatability, nonces)

Freshness

- Sometimes need a value which is guaranteed not to happen before
  - Do not care about unpredictability
  - Just do not want to reuse an old one
- Solution: keep a counter and use 1,2,...
  - Problem: requires state (predictability OK !)
- Stateless solution? Yes, pick at random
  - If pick from $\{0,1\}^K$ at most q times, then Pr[repeat somevalue] < $q^2 2^{-K}$ (birthday bound)
  - High Entropy also suffices ! (~ $q^2 2^{-H(X)}$)

Nonces and Their Applications

- Nonce = a value that "never" repeats
- Why do we care?
  1. Freshness "in time" (e.g., key exchange)
  2. Freshness of input to a block cipher or a pseudorandom function (describe later)
- Applications: key establishment (1, now), symmetric encryption, authentication (2, later), …

Example: Key Establishment

- Say, Alice and Bob know their public keys and want to establish a session key
- Simple solution: A picks K at random and sends $c = E_B(K, \text{"Alice"})$, $\sigma = Sig_A(c, \text{"Bob"})$
- <u>Problem</u>: after K is gone, Eve might learn it and reuse (c, σ), establishing a fake key with Bob
  - Replay attack !
- Solution: use a nonce R
  - B sends $(R, Sig_B(R))$
  - A replies with $c = E_B(K, \text{"Alice"})$, $\sigma = Sig_A(c, R, \text{"Bob"})$
  - Ensures Eve can't use old R with Bob
  - Privacy of R not important, as long as B doesn't reuse

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)
- Freshness (non-repeatability, nonces)
- Noise (confusing attacker)
  - Add noise to data to maintain "global features", but hide individual information
  - Mainly used for "database sanitization"
  - Recent research area (differential privacy)

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)
- Freshness (non-repeatability, nonces)
- Noise (confusing attacker)
- Efficiency ! (e.g., primality testing)
  - Could be that randomness is not inherently needed, but can speed things up!

Primality testing

- Want to know if p is prime?
  - Only recently know how to do "moderately efficiently" (K^6 best??) & deterministically
- Still, much faster to do probabilistically!
- Recall, if p-prime, $z^{p-1} = 1 \bmod p$
- Which z to test?
  - all z: exponential time ☹
  - z = 2: not bad, but many counter-examples
  - random z: "almost" works, minor fix needed
  - get famous Miller-Rabin test

Batch Verification of RSA

- Assume need to verify many (t) RSA sigs (m_i, σ_i), where $\sigma_i^e = H(m_i) \bmod n$
  - Naive solution: t exponentiations
- Idea: for any subset I of $\{1 \ldots t\}$, let
  - $M_I = \prod_{i \in I} H(m_i) \bmod n$, $\sigma_I = \prod_{i \in I} \sigma_i \bmod n$
  - Then $\sigma_I^e = M_I \bmod n$, for any I
- Pick random I and check above equation (1.5 exp)
  - If there exists a bad signature, detect w/pr $\frac{1}{2}$!
- Now repeat several (say 80) times:
  - for large t benefit outperforms the cost !

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)
- Freshness (non-repeatability, nonces)
- Noise (confusing attacker)
- Efficiency ! (e.g., primality testing)
- Probabilistically Checkable Proofs
  - Includes zero-knowledge proofs…

Proofs

- Prover P wants to prove to Verifier V some statement S is true
- Witness of S: string w s.t. V can check S is true using w
  - Ex.: S = "Second bit of Dlog(y) is 0", then w = Dlog(y). Test by seeing $w_2=0$ and $g^w = y$
- NP = class of problems where each true statement has a witness, and false statements do not have any witnesses
  - Note, witness might be hard to find, but always easy to check ! (big question: $P \neq NP$?)

Some Questions

- P can always convince V by sending w
- Question 1 (orthogonal to us, but nice!): If P is unbounded, can we convince poly-time V in problems outside of NP?
  - Yes, can do anything in polynomial space !
  - Randomness essential (else "stuck" with NP)
  - Unpredictability enough [DOPS04]
  - Won't give example (although fascinating!), since in practice no unbounded P

Short Proof?

- Questions 2: if V is OK to be "fooled" with tiny probability, can we send significantly shorter string than w?
- Batch verifier for RSA: could be viewed as P proving "I know all t signatures" without sending all of them !
- Don't care about leaking witnesses (yet!), only efficiency

Short Proof?

- Remarkable (theoretical result): any NP statement can be proven with "polylog" communication (under some assumptions)
  - Again, randomness essential here !
- In fact, P can write a moderately long (poly(n)-bit) "special proof" w' s.t. V can check correctness of w' w.r.t. S using:
  - Using O(log n) random bits
  - Reading CONSTANT number of bits of w'
  - Having 99.9% assurance he was not fooled
- Celebrated result, but very impractical ☹

Hiding the Witness?

- Questions 3: (most crypto related) Can P prove S to V s.t. (1) V is convinced; yet (2) V does not learn the witness w?
- Useful for a variety of different reasons
- Ex. 1: identification schemes
  - P proves knows SK corresponding to PK
  - Don't send SK as then V can impersonate P
  - Still leaked partial knowledge (e.g. some signatures V can't compute), just not enough to actively impersonate V

Signature or Encryption?

- Sig certainly leaks signature of new values Sig(R), which V can't get
- Enc actually doesn't leak that much…
- V expects to get Dec(c) = R, just wants to get convinced P can produce it too !
- If V "knew" P was going to pass, could have simulate the entire proof !
  - Zero-knowledge proof, nothing is leaked !
  - Well… almost. What if V asks Dec("bad c")??

Zero-Knowledge proofs

- Roughly: whatever V learned from talking to P (beyond the validity of assertion), V can "simulate" on its own!
- ZK Proofs: concentrate on statements which could be true or false (decision)
  - Ex.: $\mathrm{msb}(\mathrm{dlog}(y)_2)=0$
- ZK Proofs of Knowledge: prove that P *knows* something he claims, without leaking any info about it !
- Arguments: P is efficient using the witness w

Big Result

- Under mild assumption (OWF exist), any NP statement has a ZK Proof and ZKPoK
  - Very important result
  - Generic proof is inefficient, but efficient solutions exist for many useful languages!
  - Generic proof + all protocols use randomness in a totally crucial way (e.g., for challenges, blinding and commitments !)

Ex: ZKPoK of Discrete Log

- Common input $y = g^x$
- P proves knowledge of x
  - P to V: pick random $r \in \{1..p-1\}$ and send "commitment" $R = g^r$
  - V to P: send random $c \in \{1..p-1\}$
  - P to V: send $s = r + cx \bmod (p-1)$
  - V: check that $g^s = R\, y^c$
- Very useful in many-many apps!

Security?

- Why PoK?
  - if P responds to $c \neq c'$ with same R, then from (s, s', c, c') can solve for $x = (s-s')/(c-c')$
  - So V is "really convinced" P knows x !
- Why (honest verifier) ZK?
  - V can "fake" conversation with P, for any c
  - Recall, only need (R,c,s) s.t. $g^s = R\, y^c \bmod p$
  - Pick random s and set $R = g^s / y^c \bmod p$
  - Easy to see same distribution on (R,c,s)
- Secure as "real" P commits to R before c

Randomness in ZK Proofs?

- Essential for the verifier !
  - Otherwise P can predict all the responses and really amounts to normal "NP"-proof, which is not ZK
  - Is unpredictable randomness enough? (later)
- For many naturally occurring problems essential for the prover as well to achieve ZK (e.g. "public-coin proofs" like the DL example)

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)
- Freshness (non-repeatability, nonces)
- Noise (confusing attacker)
- Efficiency ! (e.g., primality testing)
- Probabilistically Checkable Proofs
- "Pseudorandomness" & "Extraction" !!!

Pseudorandomness

- R is pseudorandom (given Y) if hard to distinguish R from a truly uniform, random string (even given Y)
- Information-theoretic: R is random
- Computational: even though R is certainly not random, it "looks so" to a computationally bonded attacker
- Decisional Diffie-Hellman Assumption:
  - for random x,y,z have

$$\langle g, g^x, g^y, g^{xy} \rangle \approx \langle g, g^x, g^y, g^z \rangle$$

DDH and its Applications

- False in standard Z_p !
  - $\mathrm{lsb}(g^{xy}) = 0$ w/pr $\frac{3}{4}$, $\mathrm{lsb}(g^z) = 0$ w/pr $\frac{1}{2}$
  - Seems true in prime order subgroup of Z_p
  - Despite the fact that g^{xy} is *uniquely determined* by g, g^x, g^y
  - Seems important that x,y,z random
  - Much stronger assumption than DL (or CDH)!
- Many applications: DH key exchange, ElGamal Encryption, Cramer-Shoup encryption, algebraic "PRF" (see later),…

DH Key Exchange from DDH

- Alice and Bob do not share anything. Want to get a key by public discussion, s.t. secure against eavesdropper Eve
- Alice: $x \to$ random, $A = g^x$, send A to Bob
- Bob: $y \to$ random, $B = g^y$, send B to Alice
- Alice: compute $K = B^x = g^{xy}$
- Bob: compute $K = A^y = g^{xy}$
- Eve: g^{xy} looks like g^z given g, g^x, g^y

Pseudorandomness

- True randomness is expensive, hard to get, store, generate
- PR approach: start with small amount of true randomness & get more randomness which is equally good for applications !
  - DDH: $g, x, y \Rightarrow g, g^x, g^y, g^{xy}$ (from 3k -> 4k)
- Does not eliminate true randomness
- Reduces its size at the expense of (strong?) computational assumptions

Relation to Extractors

- More later, but extractors start with imperfect randomness, and try to extract nearly perfect one
  - Typically extract statistically random stuff (no computational assumptions)
  - Sometimes do not use any additional true randomness (but very limited use)
  - Sometimes use a "little" true randomness, but extract "much more" using the imperfect source "instead of" computational assumption

Main PR Primitives

- PR Generator (PRG)
  - Length increasing function G (say $k \to n$) s.t.
  - $G(U_k) \approx U_n$, where U_t - uniform on t bits
  - DDH more or less gives (a slow) PRG
- PR Function (PRF) family
  - $F = \{f_s \mid s \in \{0,1\}^k\}$ indexed by "short" key s
  - For random s, $f_s \approx$ truly random function (i.e., one with random output for every input)
  - Say, $f_s:\{0,1\}^k \to \{0,1\}$. Compress $2^k \to k$ bits !
- PR Permutation (PRP) family
  - $P = \{(\pi_s, \pi_s^{-1}) \mid s \in \{0,1\}^k\}$ - each π_s invertible!
  - For random s, $(\pi_s, \pi_s^{-1}) \approx$ truly random (g,g^{-1})

Applications of PRGs

- Beat Shannon bound on key length for one-time encryption:
  - $\mathrm{Enc}_s(M) = M \oplus G(s)$, here $|M| \gg |s|$
- Stream Ciphers: "stateful" PRGs

$$G(s_t) \rightarrow R_t,\ s_{t+1}$$

  - Give stateful sequence of OTPs
- Hybrid public-key encryption:

$$\mathrm{Enc}_{PK}'(M) = \langle\ \mathrm{Enc}_{PK}(s)\ ,\ m \oplus G(s)\ \rangle$$

  - Reduces PKE of long messages to short

Applications of PRFs/PRPs

- PRFs
  - Much easier stateful cipher: $f_s(1),\ldots f_s(t),\ldots$
  - Message authentication codes
  - Modes of operations for encryption (e.g., OFB, CFB, counter, XOR)
  - Repeated generation of same randomness!
  - Huge number of other applications
  - Essentially, f_s(nonce) is a new OTP !
- PRPs
  - PRP is a length-preserving PRF, so many of the above applications work here as well
  - Plus unique ones where inverse needed (CBC)

Example: Encryption

- Idea 1 : use PRP, $Enc_s(m) = \pi_s(m)$
  - Problem: Enc(m) always the same !
  - Cannot encrypt repeated values from small space ({sell,buy})
- Moral: repeated encryption of the same message should be different
  - Either update secret key (stateful ☹)
  - Or must be probabilistic
  - *Latter only option in the public key setting* !

Example: Encryption

- In symmetric-key setting, nonce suffices
  - $\mathrm{Enc}(m) = f_s(\mathrm{nonce}) \oplus M$
  - many ways to extend to multiple blocks, get OFB, CFB, XOR, counter
- With PRPs, can also use CBC
  - $\mathrm{Enc}(m) = \pi_s(\mathrm{nonce} \oplus M)$
- CBC *not secure* with counter, need unpredictable nonce (like random !)
- **Punchline**: "convenient" encryption must use randomness **both** for keys and per every invocation !

Relation to Unpredictability

- X is unpredictable (given Y) if hard to compute X (given Y)
  - Only makes sense in "probabilistic sense"
- Could be information-theoretic
  - Random challenge R (trivial)
  - Does not inherently require true randomness
  - High entropy necessary and sufficient
- Could be computational
  - Ex.: discrete log assumption
  - Given (p, g, g^x mod p), hard to compute x, even though x is "mathematically unique"

Aside: Comparison

- Although sampling unpredictable value (i.e., challenge) does not require true randomness, most computational unpredictability assumptions need it !
  - Ex: for discrete log, need to perfectly sample p,g,x to claim x is unpredictable
  - Can state for imperfect p,g,x, but dangerous
- In general, many differences between i.t. and computational unpredictability (stay tuned)

Back to Unpredictability

- Backbone of (computational) crypto
- Most natural assumptions (factoring, discrete log, RSA) says something is unpredictable given other info
  - Would like to avoid assuming PR if we can !
- Especially useful (i.e., sufficient) for authentication applications
  - Secure signature: sig(m) is unpredictable even given $sig(m_1)\ldots sig(m_k)$ for any $m_i \neq m$

Relation to Privacy

- Theoretically OK to leak partial info, as long as "all of" X is still hard
  - Ex: lsb(x) easy from g^x mod p, OK to leak signature of "old/unimportant" messages
- Compare to privacy apps, where cannot leak any partial info
- <u>Question</u>: is having unpredictability enough for achieving privacy (i.e., pseudorandomness)?
  - Depends on whether can sample uniform bits !

Relation to Privacy

- Beautiful BIG result [Goldreich-Levin]:
  - Assume X is unpredictable to attacker
  - Assume r is truly random but known
  - Then $X \cdot r \pmod 2$ looks random to attacker: given r, hard to guess $X \cdot r$ w/pr. > 51% !
- Generically converts UP to PR
  - Huge theoretical result (still not optimal !)
- <u>Example</u>: Alice and Bob share UP value X and want to share a PR bit
  - Alice picks random r and sends it to Bob in "the clear". Both agree on $b = X \cdot r \pmod 2$

Relation to Privacy

- Can view as a "computational extractor" !
- However, assumes true randomness r
- A lot of my work: what if cannot sample r?
  - E.g., only have unpredictable r's…
- Is UP still enough? (my work: likely NO)
- To what extent can we base cryptography on imperfect randomness ??
- Exciting, rapidly developing area !
  - starting point for this course…

Reasons for Randomness

- Key Generation
- Privacy: masking, blinding, hiding, re-randomizing
- Unpredictability (random challenges)
- Freshness (non-repeatability, nonces)
- Noise (confusing attacker)
- Efficiency ! (e.g., primality testing)
- Probabilistically Checkable Proofs
- "Pseudo-randomness" & "Extraction" !!!

Main Applications

- Encryption
- Message authentication, fingerprinting
- Secret sharing, AONTs
- Commitment Schemes
- Key Exchange
- Identification Schemes
- Zero-Knowledge Proofs
- Blinding, Anonymity, Privacy, …
- "All together" (sample e-cash application)

E-cash

Simple payment protocol:

- Sign a document transferring money from your account to another account
- This document goes to your bank
- The bank verifies that this is not a copy of a previous check
- The bank checks your balance
- The bank transfers the sum

Problems:

- Requires online access to the bank (to prevent reusage)
- Expensive.
- The transaction is traceable (namely, the bank knows about the transaction between you and Alice).

First attempt

Withdrawal

- User gets bank signature on {I am a $100 bill, #1234}
- Bank deducts $100 from user's account

Payment

- User gives the signature to a merchant
- Merchant verifies the signature, and checks online with the bank to verify that this is the first time that it is used.

Problems:

- As before, online access to the bank, and lack of anonymity.

Advantage:

- The bank doesn't have to check online whether there is money in the user's account.

Anonymous cash via blind signatures

- The bank signs the bill without seeing it (e. g. like signing on a carbon paper)
- Can use RSA Blind signatures did earlier!
- RSA signature: $H(m)^{1/e}$ mod n
- Blind RSA signature:
  - Alice: sends Bob (r^e H(m)) mod n, where r is a random
  - Bob: computes $(r^e H(m))^{1/e} = r\, H(m)^{1/e}$ mod n, and sends to Alice.
  - Alice divides by r and computes Sig(m) = $H(m)^{1/e}$ mod n
- Problem: Alice can get Bob to sign anything, as Bob does not know what he is signing.

Enabling the bank to verify the signed value

- Use "cut and choose" protocol
- Suppose Alice wants to sign a $20 bill.
  - She prepares 100 different $20 bills for blind signature, and sends them to the Bank (Bob).
  - The bank chooses 99 of them at random and asks Alice unblind them (divide by the corresponding r values).
  - It verifies that they are all $20 bills.
  - The bank blindly signs the remaining bill and gives it to Alice.
- If Alice tries to cheat she is caught with probability 99/ 100.
- 100 can be replaced by any parameter k.
- We would have preferred an exponentially small cheating probability.

Exponentially small cheating probability

- Define that a $20 bill is valid if it is the e-th root of the multiplication of 50 values of the form H (x) , (H is one-way) and the owner of the bill can present all 50 x values.
- The withdrawal protocol:
  - Alice sends to the Bank z_1 , z_2 , …, z_{100} (where $z_i = r_i^e \cdot H(x_i)$).
  - Bank asks Alice to reveal random ½ of the values $z_i = r_i^e \cdot H(x_i)$.
  - Bank verifies them and extracts the e-th root of the multiplication of all the other 50 values.
- Payment: Alice sends the signed bill and reveals the 50 preimage values. The merchant sends them to the bank which verifies that they haven't been used before.
- Alice can only cheat if she guesses the 50 locations in which she will be asked to unblind the z_i's, which happens with probability ~2^{-100}.

Online vs. offline digital cash

- We solved the anonymity problem, while verifying that Alice can only get signatures on bills of the right value
- The bills can still be duplicated
  - Merchants must check with the bank whenever they get a new bill, to verify that it wasn't used before.
- A new idea:
  - During the payment protocol the user is forced to encode a random identity string (RIS) into the bill
  - If the bill is used twice, the RIS reveals the user's identity and she loses her anonymity.

Offline digital cash

Withdrawal protocol:

- Alice prepares 100 bills of the form
  - {I am a \$20 bill, #1234, $y_1, y_1', y_2, y_2', \ldots, y_k, y_k'$}
  - S.t. for all i, $y_i = H(x_i)$, $y_i' = H(x_i')$, $x_i \oplus x_i'$ = Alice's id, where H() is a "good" hash function and x_i random
- Alice blinds these bills and sends to the bank.
- The bank asks her to unblind 99 bills and show their x_i and x_i' values, and checks their validity. (Alternatively, as in the previous example, Alice can do a check which fails with only an exponential probability.)
- The bank signs the remaining blinded bill.

Offline digital cash

Payment protocol:

- Alice gives a signed bill to the vendor
  - {I am a $20 bill, #1234, $y_1, y_1', y_2, y_2', \ldots, y_k, y_k'$}
- The vendor verifies the signature, and if valid sends to Alice a random bit string $b = b_1 b_2 \ldots b_k$ of length k.
- For all i, if $b_i=0$ Alice returns x_i, otherwise ($b_i=1$) she returns x_i'
- The vendor checks that $y_i = H(x_i)$ or $y_i' = H(x_i')$ (depending on b_i). If this check is successful it accepts the bill.
- Note that Alice's identity is kept secret!
- Also, the merchant does not need to contact the bank during the payment protocol.

Offline digital cash

- The merchant must deposit the bill in the bank. It cannot use the bill to pay someone else.
  - Because it can't answer challenges b* different from the challenge b it sent to Alice.
- How can the bank detect double spenders?
  - Suppose two merchants M and M* receive same bill
  - With very high probability, they send different queries b, b*
  - Suppose $b_i = 0$, $b_i^* = 1$. Then M receives x_i and M* receives x_i'.
  - When they deposit the bills the bank receives both x_i and x_i', and can compute $x_i \oplus x_i'$ =Alice's id.

Usage of Randomness

Several very different uses !

1. To generate signing/verification key (SK and PK)
2. To blind RSA signatures (random r)
3. To perform cut-and-choose proofs (random 1/2 blindings to open)
4. To randomly open 1-of-2 values of x_i (b)
5. To prevent double-spending (split randomly $x_i \oplus x_i'$)