Pseudorandom Generators and The Pseudo One Time Pad

Lecture 6

Review – Security

Perfect secrecy and indistinguishability

Perfect secrecy and indistinguishability

- Requires that *absolutely no information* about the plaintext is leaked, even to eavesdroppers *with unlimited computational power*

Perfect secrecy and indistinguishability

- Requires that *absolutely no information* about the plaintext is leaked, even to eavesdroppers *with unlimited computational power*

- Π is perfectly indistinguishable $\Leftrightarrow$ Π is perfectly secret

Perfect secrecy

- Encryption scheme (Gen, Enc, Dec) with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is *perfectly secret* if for every distribution over $\mathcal{M}$, every m $\in$ $\mathcal{M}$, and every c $\in$ $\mathcal{C}$ with Pr[C=c] > 0, it holds that

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

Perfect indistinguishability

- Let Π=(Gen, Enc, Dec) be an encryption scheme with message space $\mathcal{M}$, and A an adversary

Perfect indistinguishability

- Let Π=(Gen, Enc, Dec) be an encryption scheme with message space $\mathcal{M}$, and A an adversary
- Define a randomized exp't $\text{PrivK}_{A,\Pi}$:
  1. A outputs $m_0, m_1 \in \mathcal{M}$
  2. $k \leftarrow \text{Gen}$, $b \leftarrow \{0,1\}$, $c \leftarrow \text{Enc}_k(m_b)$
  3. $b' \leftarrow A(c)$

  Adversary A *succeeds* if b = b', and we say the experiment evaluates to 1 in this case

Perfect indistinguishability

- Π is *perfectly indistinguishable* if for all attackers (algorithms) A, it holds that

$$\Pr[\mathrm{PrivK}_{A,\Pi} = 1] = ½$$

Perfect indistinguishability

- Π is *perfectly indistinguishable* if for all attackers (algorithms) A, it holds that

$$\Pr[\text{PrivK}_{A,\Pi} = 1] = \frac{1}{2}$$

- Π is perfectly indistinguishable => Π is perfectly secret

Computational secrecy

Computational secrecy

- Would be ok if a scheme leaked information *with tiny probability* to eavesdroppers *with bounded computational resources*

Computational secrecy

- Would be ok if a scheme leaked information *with tiny probability* to eavesdroppers *with bounded computational resources*
- Relax perfect secrecy by
  - Allowing security to “fail” with *negligible* probability
  - Restricting attention to PPT attackers

Asymptotic security

- Introduce *security parameter* n

Asymptotic security

- Introduce *security parameter* n
  - For now, think of n as the key length
  - Chosen by honest parties when they generate/share key
    - Allows users to tailor the security level
  - Known by adversary

Asymptotic security

- Introduce *security parameter* n
  - For now, think of n as the key length
  - Chosen by honest parties when they generate/share key
    - Allows users to tailor the security level
  - Known by adversary

- Measure running times of all parties, and the success probability of the adversary, as functions of n

Computational indistinguishability (asymptotic)

Computational indistinguishability (asymptotic)

- Computational indistinguishability:
  - Security may fail with probability *negligible in n*
  - Restrict attention to attackers running in time (at most) *polynomial in n*

Computational indistinguishability (asymptotic)

- Computational indistinguishability:
  - Security may fail with probability *negligible in n*
  - Restrict attention to attackers running in time (at most) *polynomial in n*

- A scheme is secure: if for every probabilistic polynomial-time adversary A carrying out an attack of some specifed type, the probability that A succeeds in this attack is negligible.

Definitions

- A function f: $Z^+ \rightarrow Z^+$ is *polynomial* if there exists $\{c_i\}$ such that f(n) < $\sum_i \; c_i \, n^i$

Definitions

- A function f: $Z^+ \rightarrow Z^+$ is *polynomial* if there exists $\{c_i\}$ such that f(n) < $\sum_i \; c_i \, n^i$

- A function f: $Z^+ \rightarrow [0,1]$ is *negligible* if for every polynomial p it holds that f(n) < 1/p(n) for large enough n

Example

- The following functions are all negligible :
  - $\frac{1}{2^n}$
  - $2^{-\sqrt{n}}$
  - $n^{-\log(n)}$
  - $\frac{f(n)}{2^n}$ where f(n) is a polynomial

Closure Properties

- Let n_1 and n_2 be negligible functions.

Closure Properties

- Let n_1 and n_2 be negligible functions.
  - 1. Then n_1(n)+ n_1(n) is negligible.

Closure Properties

- Let n_1 and n_2 be negligible functions.
  - 1. Then n_1(n)+ n_1(n) is negligible.
  - 2. For any positive polynomial p, the function p(n) $*$ n_1(n) is negligible.

Example

- The function $\frac{f(n)}{2^n}$ is negligible where f(n) is a positive polynomial.

Example

- The function $\frac{f(n)}{2^n}$ is negligible where f(n) is a positive polynomial.

- Proof
  - 1. $\frac{1}{2^n}$ is negligible

Example

- The function $\frac{f(n)}{2^n}$ is negligible where f(n) is a positive polynomial.

- Proof
  - 1. $\frac{1}{2^n}$ is negligible
  - 2. f(n) is a polynomial, hence $\frac{f(n)}{2^n}$ $is\ negl$

Deterministic/Probabilistic

Deterministic/Probabilistic

- An algorithm A is deterministic if for any input c, the output A(c) = A(c) for every application of A.

Deterministic/Probabilistic

- An algorithm A is deterministic if for any input c, the output A(c) = A(c) for every application of A.

- An algorithm A is probabilistic if for any input c, the output A(c) need not be equal to A(c) for every application of A.

Deterministic/Probabilistic

- A probabilistic algorithm with running time p and an input of length n, yields an unbiased random bits string of length p(n) where each bit is independently equal to 1 with probability 1/2 and 0 with probability 1/2.

(Re)defining encryption

- A *private-key encryption scheme* is defined by three PPT algorithms (Gen, Enc, Dec):
  - Gen: takes as input 1^n; outputs k. (Assume $|k| \geq n$.)
  - Enc: takes as input a key k and message $m \in \{0,1\}^*$; outputs ciphertext c

$$c \leftarrow \mathrm{Enc}_k(m)$$

  - Dec: takes key k and ciphertext c as input; outputs a message m or "error"

Computational indistinguishability (asymptotic version)

- Fix a scheme Π and some adversary A
- Define a randomized exp't $\text{PrivK}_{A,\Pi}(n)$:
  1. $A(1^n)$ outputs $m_0, m_1 \in \{0,1\}^*$ of equal length
  2. $k \leftarrow \text{Gen}(1^n)$, $b \leftarrow \{0,1\}$, $c \leftarrow \text{Enc}_k(m_b)$
  3. $b' \leftarrow A(c)$

  Adversary A *succeeds* if b = b', and we say the experiment evaluates to 1 in this case

Computational indistinguishability (asymptotic version)

- Π is *computationally indistinguishable* if for all PPT attackers A, there is a negligible function ε such that

$$\Pr[\mathrm{PrivK}_{A,\Pi}(n) = 1] \leq \tfrac{1}{2} + \varepsilon(n)$$

Pseudorandomness

Pseudorandomness

Pseudorandomness

- Important building block for computationally secure encryption

Pseudorandomness

- Important building block for computationally secure encryption
- Important concept in cryptography

What does “random” mean?

What does “random” mean?

- What does “uniform” mean?

What does “random” mean?

- What does “uniform” mean?
- Which of the following is a uniform string?

What does "random" mean?

- What does "uniform" mean?
- Which of the following is a uniform string?
  - 0101010101010101
  - 0010111011100110
  - 0000000000000000

What does “random” mean?

- What does “uniform” mean?
- Which of the following is a uniform string?
  - 0101010101010101
  - 0010111011100110
  - 0000000000000000
- If we generate a uniform 16-bit string, each of the above occurs with probability 2^{-16}

What does “uniform” mean?

- “Uniformity” is not a property of a *string*, but a property of a *distribution*

- A distribution on *n*-bit strings is a function $D: \{0,1\}^n \rightarrow [0,1]$ such that $\Sigma_x D(x) = 1$
  - The *uniform* distribution on *n*-bit strings, denoted U_n, assigns probability 2^{-n} to every $x \in \{0,1\}^n$

What does “uniform” mean?

- “Uniformity” is not a property of a *string*, but a property of a *distribution*

- A distribution on *n*-bit strings is a function $D: \{0,1\}^n \to [0,1]$ such that $\Sigma_x D(x) = 1$
  - The *uniform* distribution on *n*-bit strings, denoted U_n, assigns probability 2^{-n} to every $x \in \{0,1\}^n$

Example

- Binary Strings of Length 2: pr(b) = 1/4

{01,10,00,11}

Example

- Binary Strings of Length 2: pr(b) = 1/4

{01,10,00,11}

- Binary Strings of Length 4: pr(b) = $\left(\frac{1}{2}\right)^4$

{ 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111}

What does “pseudorandom” mean?

What does “pseudorandom” mean?

- Informal: cannot be distinguished from uniform (i.e., random)

What does “pseudorandom” mean?

- Informal: cannot be distinguished from uniform (i.e., random)
- Which of the following is pseudorandom?
  - 0101010101010101
  - 0010111011100110
  - 0000000000000000

What does “pseudorandom” mean?

- Informal: cannot be distinguished from uniform (i.e., random)
- Which of the following is pseudorandom?
  - 0101010101010101
  - 0010111011100110
  - 0000000000000000
- Pseudorandomness is a property of a *distribution*, not a *string*

Pseudorandomness (concrete)

- $x \leftarrow D$ means "sample x according to D"

- Let D be a distribution on *n*-bit strings

- D is (t, ε)-pseudorandom if for all A running in time at most t,

$$| \Pr_{x \leftarrow D}[A(x)=1] - \Pr_{x \leftarrow U_p}[A(x)=1] | \leq \varepsilon$$

Pseudorandomness (asymptotic)

- Security parameter *n*, polynomial *p*
- Let D_n be a distribution over *p(n)*-bit strings
- Pseudorandomness is a property of a *sequence* of distributions $\{D_n\} = \{D_1, D_2, \ldots\}$

Pseudorandomness (asymptotic)

- $\{D_n\}$ is *pseudorandom* if for all probabilistic, polynomial-time distinguishers A, there is a negligible function ε such that

$$\left| \Pr_{x \leftarrow D_n}[A(x)=1] - \Pr_{x \leftarrow U_{p(n)}}[A(x)=1] \right| \leq \varepsilon(n)$$

Pseudorandom generators (PRGs)

Pseudorandom generators (PRGs)

- A PRG is an efficient, deterministic algorithm that expands a *short, uniform seed* into a *longer, pseudorandom* output

Pseudorandom generators (PRGs)

- A PRG is an efficient, deterministic algorithm that expands a *short, uniform seed* into a *longer, pseudorandom* output
  - Useful whenever you have a “small” number of true random bits, and want lots of “random-looking” bits

Pseudorandom generators (PRGs)

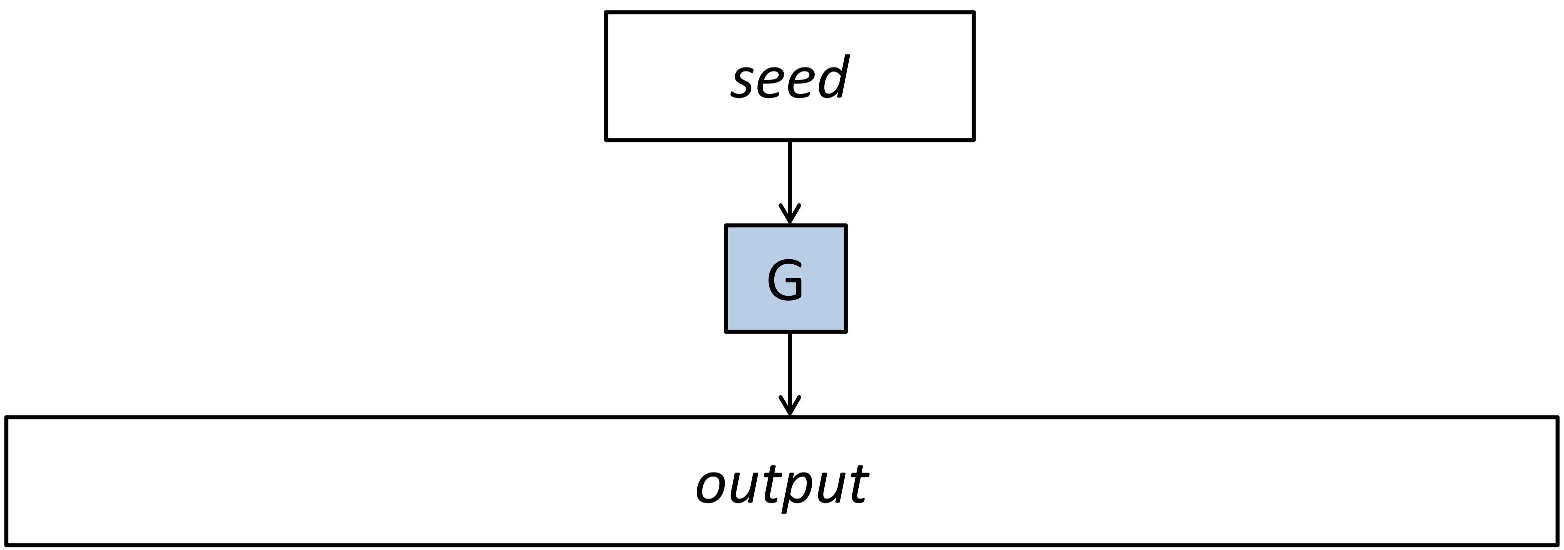
- A PRG is an efficient, deterministic algorithm that expands a *short, uniform seed* into a *longer, pseudorandom* output
  – Useful whenever you have a “small” number of true random bits, and want lots of “random-looking” bits

PRGs

- Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$

PRGs

- Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$



PRGs

- Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$

PRGs

- Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$
- G defines a sequence of distributions!

PRGs

- Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$
- G defines a sequence of distributions!
  - D_n = the distribution on p(n)-bit strings defined by choosing $x \leftarrow U_n$ and outputting G(x)

PRGs

- Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$
- G defines a sequence of distributions!
  - D_n = the distribution on p(n)-bit strings defined by choosing $x \leftarrow U_n$ and outputting G(x)
  - $Pr_{D_n}[y] = |\{x : G(x)=y\}|/2^n$
  - Note that most y occur with probability 0
    - I.e., D_n is far from uniform

PRGs

- G is a PRG iff $\{D_n\}$ is pseudorandom

- I.e., for all efficient distinguishers A, there is a negligible function ε such that

  $$| \Pr_{x \leftarrow U_n}[A(G(x))=1] - \Pr_{y \leftarrow U_{p(n)}}[A(y)=1] | \leq \varepsilon(n)$$

- I.e., no efficient A can distinguish whether it is given G(x) (for uniform x) or a uniform string y!

Example

Do PRGs exist?

- We don't know…
  - Would imply P $\neq$ NP
- We will *assume* certain algorithms are PRGs
  - Recall the 3 principles of modern crypto…
  - This is what is done in practice
  - We will return to this later in the course
- Can *construct* PRGs from weaker assumptions
  - For details, see Chapter 7

Where things stand

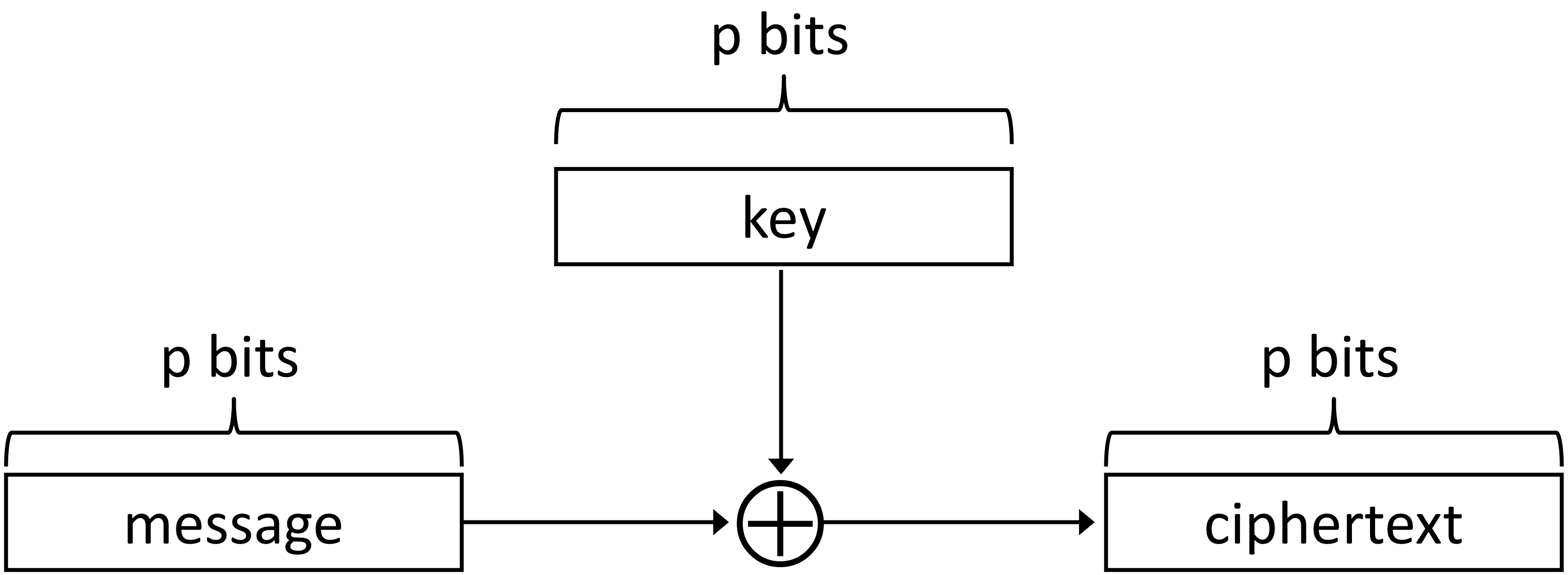
- We saw that there are some inherent limitations if we want perfect secrecy
  - In particular, key must be as long as the message

- We defined computational secrecy, a relaxed notion of security

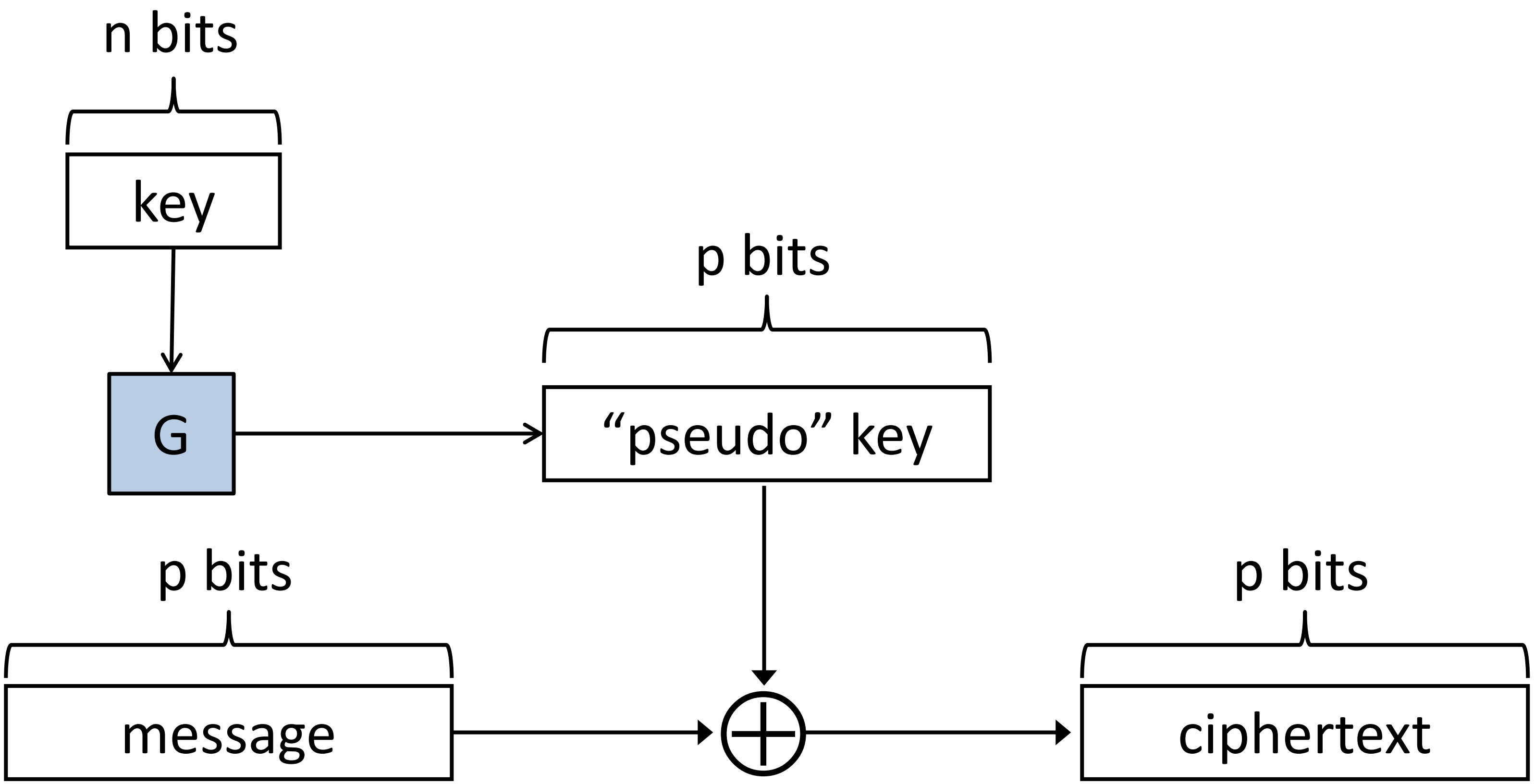
- Can we overcome prior limitations?

Pseudo-one time pad

Recall: one-time pad



“Pseudo” one-time pad



Pseudo one-time pad

- Let G be a deterministic algorithm, with $|G(k)| = p(|k|)$
- $Gen(1^n)$: output uniform n-bit key k
  - Security parameter n $\Rightarrow$ message space $\{0,1\}^{p(n)}$
- $Enc_k(m)$: output $G(k) \oplus m$
- $Dec_k(c)$: output $G(k) \oplus c$

- Correctness is obvious…

Security of pseudo-OTP?

- Would like to be able to *prove* security
  - Based on the *assumption* that G is a PRG

Definitions, proofs, and assumptions

- We've *defined* computational secrecy
- Our goal is to *prove* that the pseudo OTP meets that definition
- We cannot prove this unconditionally
  - Beyond our current techniques…
  - Anyway, security clearly depends on G
- *Can* prove security based on *the assumption* that G is a pseudorandom generator

Security theorem

- If G is a pseudorandom generator, then the pseudo one-time pad Π is secure (i.e., computationally indistinguishable)

Stepping back…

- *Proof* that the pseudo OTP is secure…
- …with some caveats
  - Assuming G is a pseudorandom generator
  - Relative to our definition

- The *only* ways the scheme can be broken are:
  - If a weakness is found in G
  - If the definition isn't sufficiently strong…

Have we gained anything?

- YES: the pseudo-OTP has a key shorter than the message
  - n bits vs. p(n) bits
- The fact that the parties *internally* generate a p(n)-bit temporary string to encrypt/decrypt is **irrelevant**
  - The *key* is what the parties share *in advance*
  - Parties do not store the p(n)-bit temporary value