Introduction to Modern Cryptography [2]
Lecture 9

December 27, 2016

Instructor: Benny Chor
Teaching Assistant: Orit Moskovich

School of Computer Science
Tel-Aviv University

Fall Semester, 2016–17
Tuesday 12:00–15:00
Venue: Meron Hall, Trubowicz 102 (faculty of Law)

Course site: http://tau-crypto-f16.wikidot.com/

[2] much of this presentation is taken from Benny Applebaum

Lecture 9: Plan

- Digital Signatures (mostly a reminder).
- Diffie Hellman signatures paradigm and its shortcomings.
- ElGamal signature scheme.
- The hash and decrypt signature paradigm.
- Other (mostly theoretical) signatures paradigm.

- Introduction to zero knowledge proofs.

Signatures

http://lacourphoto.net/uploaded_images/signatures1-770492.jpg

Digital Signature Schemes (reminder)

- Relate an individual, through a digital string, to a document.
- Would like to achieve all features of hand written signatures, plus more.
- For example, should be able to base difficulty of forgery on some hard computational problem, not just on ineptitude of forger.
- Diffie and Hellman were first to propose such framework.
- An implementation was first given by RSA.

Signatures vs. MACs: Important Distinction (reminder)

You surely remember message authentication codes (MACs).

- The obvious syntactic difference between signatures and MACs is that MACs require shared secret key, while the secret signature key is known to one user only.

- More importantly, semantically:
- Suppose parties A and B share the secret key K. Then $(M, MAC_K(M))$ convinces A that indeed M originated with B. But nobody else can verify this.
- By way of contrast, given a signature of A on M, it can be verified by anybody that the signature was created by A. Yet nobody else can create such signature.

Digital Signatures: Definition (reminder)

Digital signature is the public-key "analog" of MAC schemes.

- Message space $\mathcal{M}$ (usually long binary strings, e.g., $\{0,1\}^*$)
- Key-Generation algorithm: generates pairs:
  Secret signing key, $\mathtt{sk}$, and public verification key $\mathtt{vk}$
- Signing algorithm: $S_{\mathtt{sk}}(m) \mapsto \sigma$
- Typically, $\sigma \in \{0,1\}^{\ell}$ where ℓ is relatively short
- Verification algorithm: $V_{\mathtt{vk}}(m,\sigma)$ outputs "accept"/"reject"

Security: Intuitively, should be hard to forge a valid tag even after seeing many legal tags

Security (Goldwasser, Micali, Rivest, 1984)[3]

Definition: (Existential Forgery under Chosen Plaintext Attack): A Signature is (t, ε)-secure if every polynomial time bounded adversary $\mathcal{A}$ which gets the public verification key $\mathtt{vk}$ and is allowed to ask for t legal pairs $(m_1, S_{\mathtt{sk}}(m_1)), \ldots, (m_t, S_{\mathtt{sk}}(m_t))$ outputs a new valid pair (m, σ) (such that $V_{\mathtt{vk}}(m, \sigma)$ accepts) with probability at most ε.

- The probability is taken over the choice of the random keys
- Adversary can choose the documents adaptively
- The adversary succeeds even if the document being forged is "meaningless". (As it is hard to predict what has and what does not have a meaning in an unknown context, and how will the receiver react to such successful forgery.)
- Want: large t and small ε (asymptotically, both are super-polynomial, or even exponential, in the key length.)

[3] A paradoxical solution to the signature problem.

Diffie and Hellman "New Directions in Cryptography" (76)

Diffie and Hellman proposed a "textbook framework", based on any deterministic public key cryptosystem.

We will describe this framework and implementation(s), and then discuss some specific shortcomings of it.

- Let E_A be Alice's public encryption key, and let D_A be Alice's private decryption key.
- To sign the message M, Alice computes the string $y = D_A(M)$ and sends (M, y) to Bob.
- To verify this is indeed Alice's signature, Bob computes the string $x = E_A(y)$ and checks that indeed $x = M$.

Remark: We tacitly assume here that E_A and D_A commute. This usually is guaranteed as both are permutations, and each is the inverse of the other.

Diffie and Hellman "New Directions in Cryptography" (cont.)

Diffie and Hellman proposed a "textbook framework", based on any deterministic public key cryptosystem.

We will describe this framework and some implementation(s), and then discuss some specific shortcomings of it.

Intuition: Only Alice can efficiently compute $y = D_A(M)$, thus forgery should be computationally infeasible.

Question: Do you buy this argument?

Problems with "Pure" Diffie-Hellman Paradigm

- Easy to forge signatures of random messages, even without holding D_A:
- Bob picks R arbitrarily, and computes $s = E_A(R)$ using Alice public key.
- Then the pair (s, R) is a valid signature of Alice on the "message" s.
- Therefore the scheme is subject to existential forgery.
- "So what" ?

Problems with "Pure" Diffie-Hellman Paradigm, cont.

- Easy to forge signatures of random messages, even without holding D_A:
- Bob picks R arbitrarily, and computes $s = E_A(R)$ using Alice public key.
- Then the pair (s, R) is a valid signature of Alice on the "message" s.
- Therefore the scheme is subject to existential forgery.

- Specific implementations may pose additional threats.
- For example, since RSA is multiplicative, we have $D_A(M_1 M_2) = D_A(M_1) D_A(M_2)$ (products are modulo pq).

Problems Specific to RSA Implementation

- Consider specifically RSA. Being multiplicative, we have $D_A(M_1M_2) = D_A(M_1)D_A(M_2)$ (products are modulo pq).

- If M_1="`I OWE BOB $20`" and M_2 = "`100`" then under certain encodings of letters, we could get M_1M_2 ="`I OWE BOB $2000`" ...

Generic Solution: Hash First

Let (E, D) be a family of one way trapdoor permutations' pairs (instantiated by choosing a secret key by the user).

- Let E_A be Alice's public encryption key, and let D_A be Alice's private decryption key.
- Let H be an ideal hash function: It hashes all strings into a typically smaller range, and "behaves like a random function".
- The hash function H is completely public, and in particular no secret key is employed.

Generic Solution: Hash First (cont.)

- To sign the message M, Alice first hashes the message, namely computes the string $y = H(M)$.
- Then, Alice computes the string $z = D_A(y)$. She sends (M, z) to Bob.
- To verify this is indeed Alice's signature, Bob computes the string $y = E_A(z)$ and checks that indeed $y = H(M)$.

- Argue (intuitively) why this foils previous attacks.

Generic Solution: Hash First

- Under the assumption that the hash function is ideal (aka the random oracle model) and the underlying public key encryption is a trapdoor one way permutation, it has been proven that the "hash first" paradigm is secure against chosen plaintext attacks.
- However, the random oracle assumption seems to require more than just the "standard assumptions".
- Important observation: A pseudo random function with a known key is not random anymore!

- It is not known how to prove that the "hash first" paradigm is secure against chosen plaintext attacks under the assumption that the hash function is "just" collision resistant.

Signature Schemes Used in Practice

- RSA (with hashing first, *e.g.* SHA3).

- ElGamal Signature Scheme (1985), based on the intractability of discrete log in Z_p^*. Will be described shortly.
- The DSS/DSA (digital signature standard/algorithm), adopted by NIST in 1994. It is based on a modification of ElGamal signature.
- ElGamal like scheme in the different groups of elliptic curves. This will probably not be described in this course.

Signature Schemes Used in Practice

- RSA (with hashing first, *e.g.* SHA3).

- ElGamal Signature Scheme (1985), based on the intractability of discrete log in Z_p^*. Will be described shortly.
- The DSS/DSA (digital signature standard/algorithm), adopted by NIST in 1994. It is based on a modification of ElGamal signature.
- ElGamal like scheme in the different groups of elliptic curves. This will probably not be described in this course.

ElGamal Probabilistic PKC (reminder)

- Public information: A large prime p, where $p-1$ has a known factorization and a large prime factor. Recommended to take $p = 2q+1$, q is prime too, and p is 756 or 1024 bits long.
  - A multiplicative generator g of Z_p^*
  - Bob publishes p, g.
  - Bob picks $a \in [0..p-2]$ at random.
  - Bob computes and publishes $\beta = g^a \pmod p$.
- Bob's private information: a.
- Encryption: of the message m:
  - Alice picks $k \in [0..p-2]$ at random.
  - Alice computes $g^k \pmod p$, $m\beta^k \pmod p$.
  - Alice sends $E(m) = (g^k, m \cdot \beta^k)$ to Bob.
    (β^k "masks" m; k obviously is not made public).
- Decryption of $(g^k, m \cdot \beta^k) = (c_1, c_2)$:
  - Bob computes ${c_1}^a = (g^k)^a = (g^a)^k = \beta^k \pmod p$.
  - This enables Bob to compute the multiplicative inverse of $\beta^k \bmod p$, β^{-k} (even though he does not know k).
  - Bob now computes $\beta^{-k} \cdot c_2 = m$. ♠

Properties of ElGamal Public Key Cryptosystem (reminder)

- Encryption is randomized: $m \to (g^k, m\beta^k)$.
- Alice should use a new, independent k for every encryption.
- Even if same m is sent twice, different k must be used.

- Encryption takes two modular exponentiations.
- Decryption takes one modular exponentiation.
- Ciphertext, $(g^k, m\beta^k)$, is twice as long as the plaintext, m.

Properties of ElGamal Public Key Cryptosystem (reminder2)

- Cryptosystem is vulnerable to chosen ciphertext attacks.
- Given $E(m) = (c_1, c_2) = (g^k, m\beta^k)$,
- Attacker chooses a random s, computes $(c_1, s \cdot c_2) = (g^k, s \cdot m\beta^k)$
- Attacker asks for decryption of $(c_1, s \cdot c_2)$, which equals $s \cdot m$, from which m is easily recovered.

- Cryptosystem is multiplicative. Given $E(m) = (c_1, c_2) = (g^k, m\beta^k)$, $E(m') = (c_1', c_2') = (g^{k'}, m'\beta^{k'})$, can easily obtain $E(m \cdot m') = (c_1 c_1', c_2 c_2') = (g^{k+k'}, m \cdot m' \beta^{k+k'})$ (without knowing any secret information).

ElGamal Signature Scheme

- Key Generation:
  A large prime p, with $p-1$ having a known factorization and a large prime factor. Recommended to take $p = 2q + 1$, where q is also a prime, and p is 1024 bits long.
  A standard, collision resistant hash function, H.
  - Bob picks a multiplicative generator g of Z_p^*.
  - Bob picks $x \in [0..p-2]$ at random.
  - Bob computes and publishes $y = g^x \pmod{p}$.
- Bob's private key is x.
- Bob's public signature key is p, g, y.

So far it is similar to ElGamal encryption. This is not a coincidence (think of the hash and sign pardigm in the context of ElGamal's probabilistic encryption).

ElGamal Signature Scheme (2)

Signing the message M, employing randomization:

- Hash first: Let $m = H(M)$:
- Bob picks at random $k \in [0..p-2]$ which is relatively prime to $p-1$ (for $p = 2q+1$, there are $\varphi(p-1) = q-1$ such elements).
- Bob computes $r = g^k \pmod{p}$. This does not depend on m.
- Bob computes $s = (m - rx) \cdot k^{-1} \bmod (p-1)$ (***)
- Bob outputs r, s.
- Together with M, this is the signature of M.

ElGamal Signature Scheme (3)

- Bob outputs r, s.
- Together with M, this is the signature of M.

- Verification of M's signature:
- Alice computes $m = H(M)$.
- If $0 < r < p$ and $y^r r^s = g^m \pmod{p}$, Alice accepts. Otherwise she rejects.

ElGamal Signature Scheme (3)

- Bob outputs r, s.
- Together with M, this is the signature of M.

- Verification of M's signature:
- Alice computes $m = H(M)$.
- If $0 < r < p$ and $y^r r^s = g^m \pmod{p}$, Alice accepts.
  Otherwise she rejects.

- What the &%$# is going on? (this is actually not black magic.)
- By (***), $s = (m - rx) \cdot k^{-1} \mod (p-1)$.
  So $sk + rx = m \mod (p-1)$.
- By construction $r = g^k$, so $r^s = g^{ks}$.
- Since $y = g^x$, we have $y^r = g^{rx}$,
  implying $y^r r^s = g^{rx} g^{ks} = g^{sk+rx} = g^m$.

♠

ElGamal Signature Scheme: Some Properties

- Since signing is randomized, same message could have multiple, different signatures. OK to sign same message many times.
- If $k \in [0..p-2]$ is repeated in different mssgs (rather than being chosen at random), Eve can extract the private key (try this!).
- Same applies to highly dependent choices of $k_1, k_2, \ldots$.
- Can be applied in other commutative groups where discrete log is believed to be hard. For example in Elliptic curves (more efficient operations).
- Forging amounts to finding r, s such that $y^r r^s = g^m \pmod{p}$, and seems to require discrete logs (but not proved equivalent!)
- Overhead:
  - Signature: One exponentiation, can be done offline (preprocessing). A constant number of modular multiplications/gcd (cheap).
  - Verification: Three exponentiations, plus a constant number of modular multiplication.

Signature Schemes: General Remarks

- Signature schemes that deviate from the Diffie-Helman plus hashing paradigm do exist.
- For example, Goldwasser-Micali-Rivest, 1988.

- Theorem: One way functions imply signature schemes secure against (adaptively) chosen plaintext attacks!
- Constructions using one way functions are generally less practical than these using hash as a random oracle, though.

Theoretical Constructions: One-time Signatures

For some years it was not clear whether signature schemes can be constructed in the standard model.

Theorem (Lamport, GMR, Naor-Yung, Rompel, Goldreich)

If one-way functions exist then there are signature schemes which are existentially unforgeable under chosen plaintext attacks.

Theoretical Constructions: One-time Signatures

For some years it was not clear whether signature schemes can be constructed in the standard model.

Theorem (Lamport, GMR, Naor-Yung, Rompel, Goldreich)

If one-way functions exist then there are signature schemes which are existentially unforgeable under chosen plaintext attacks.

Remarks:

- Surprise: a primitive (signature) that seems related to trapdoor one way functions, can be based on just one way functions!
- The construction is not practical, but serves as an important feasibility result.
- The proof contains many useful ideas and techniques (which are also used in practice).
- We will describe a slightly weaker result, step by step.

Signing a single bit

Let $f : X \to Y$ be a one-way function.

Q: Can we sign a single bit?

Signing a single bit

Let $f : X \to Y$ be a one-way function.

Q: Can we sign a single bit?

- Secret key: $x_0, x_1 \leftarrow X$.
- Public key: $y_0 = f(x_0), y_1 = f(x_1)$.
- Sign 0 by x_0 and sign 1 by x_1.
- Verify (m, x_m) by comparing $f(x_m)$ to y_m.

Signing a single bit

Let $f : X \to Y$ be a one-way function.

Q: Can we sign a single bit?

- Secret key: $x_0, x_1 \leftarrow X$.
- Public key: $y_0 = f(x_0), y_1 = f(x_1)$.
- Sign 0 by x_0 and sign 1 by x_1.
- Verify (m, x_m) by comparing $f(x_m)$ to y_m.

Security?

Signing a single bit

Let $f : X \to Y$ be a one-way function.

Q: Can we sign a single bit?

- Secret key: $x_0, x_1 \leftarrow X$.
- Public key: $y_0 = f(x_0), y_1 = f(x_1)$.
- Sign 0 by x_0 and sign 1 by x_1.
- Verify (m, x_m) by comparing $f(x_m)$ to y_m.

Security?

Q: Can we construct a signature on n bits which remains secure if the adversary sees a single signature?

One-Time Signature for Three Bits

Let $f : X \to Y$ be a one-way function.

$y_{0,1} = f(x_{0,1})$	$y_{0,2} = f(x_{0,2})$	$y_{0,3} = f(x_{0,3})$
$y_{1,1} = f(x_{1,1})$	$y_{1,2} = f(x_{1,2})$	$y_{1,3} = f(x_{1,3})$

One-Time Signature for Three Bits

Let $f : X \to Y$ be a one-way function.

$y_{0,1} = f(x_{0,1})$	$y_{0,2} = f(x_{0,2})$	$y_{0,3} = f(x_{0,3})$
$y_{1,1} = f(x_{1,1})$	$y_{1,2} = f(x_{1,2})$	$y_{1,3} = f(x_{1,3})$

- To sign 010 reveal $x_{0,1}, x_{1,2}, x_{0,3}$.
- Given a signature of 010, can you forge a signature on a different document, say 100?
- Given signatures on 010 and 100, can you forge a signature on a new document?

Signing n-bits messages

Lamport's One-Time signature

- Let $f : X \to Y$ be a one-way function.
- Secret key: Choose $2n$ random strings $x_{b,i} \leftarrow X$ for $b \in \{0,1\}, 1 \leq i \leq n$.
- Public key: $y_{b,i} = f(x_{b,i})$ for $b \in \{0,1\}, 1 \leq i \leq n$.
- Sign a document $m = m_1, \ldots, m_n$ by $x_{m_1,1}, \ldots, x_{m_n,n}$.
- Verify a signature $x'_1, \ldots, x'_n$ on a document $m = m_1, \ldots, m_n$ by checking that $y_{m_i,i} = f(x'_i)$ for all $1 \leq i \leq n$.

Signing n-bits messages: Security

Thm: Let f be (t, ε) one-way function of complexity s (namely after seeing t pairs $y, f^{-1}(y)$ chosen adaptively, a s time adversary has prob. $\leq \varepsilon$ of inverting a new, randomly chosen z).
Then we get $(t' = t - 2ns, \varepsilon' = 2n\varepsilon)$ one-time signature.

Signing n-bits messages: Security

Thm: Let f be (t, ε) one-way function of complexity s (namely after seeing t pairs $y, f^{-1}(y)$ chosen adaptively, a s time adversary has prob. $\leq \varepsilon$ of inverting a new, randomly chosen z).
Then we get $(t' = t - 2ns, \varepsilon' = 2n\varepsilon)$ one-time signature.

That is, any t'-time adversary $\mathcal{A}$, that is allowed to ask for a signature σ' on an arbitrary document m', succeeds in forging a signature σ on a new document $m \neq m'$ with probability at most ε'.

Lamport's One-Time Signatures – Efficiency

Lamport's One-Time signature

- Let $f : X \to Y$ be a one-way function.
- Secret key: Choose $2n$ random strings $x_{b,i} \leftarrow X$ for $b \in \{0,1\}, 1 \leq i \leq n$.
- Public key: $y_{b,i} = f(x_{b,i})$ for $b \in \{0,1\}, 1 \leq i \leq n$.
- Sign a document $m = m_1, \ldots, m_n$ by $x_{m_1,1}, \ldots, x_{m_n,n}$.
- Verify a signature $x'_1, \ldots, x'_n$ on a document $m = m_1, \ldots, m_n$ by checking that $y_{m_i,i} = f(x'_i)$ for all $1 \leq i \leq n$.

Lamport's One-Time Signatures – Efficiency

Lamport's One-Time signature

- Let $f : X \to Y$ be a one-way function.
- Secret key: Choose $2n$ random strings $x_{b,i} \leftarrow X$ for $b \in \{0,1\}, 1 \leq i \leq n$.
- Public key: $y_{b,i} = f(x_{b,i})$ for $b \in \{0,1\}, 1 \leq i \leq n$.
- Sign a document $m = m_1, \ldots, m_n$ by $x_{m_1,1}, \ldots, x_{m_n,n}$.
- Verify a signature $x'_1, \ldots, x'_n$ on a document $m = m_1, \ldots, m_n$ by checking that $y_{m_i,i} = f(x'_i)$ for all $1 \leq i \leq n$.

Problems:

- Long keys (longer than the message!).
- Expensive in communication and computation.
- Only one-time security.

Signing Long Documents

Q: How to sign a document longer than the key?

Signing Long Documents

Q: How to sign a document longer than the key?

Idea: Hash, then sign the hashed value!

Short One-Time Signatures for a Long Document

Ingredients:

- $H : \{0,1\}^* \to \{0,1\}^n$ a (t, ε) collision-resistance hash function.
- (KG, S, V) a (t, ε) one-time signature for n-bit documents.

Short One-Time Signatures for a Long Document

Ingredients:

- $H : \{0,1\}^* \to \{0,1\}^n$ a (t, ε) collision-resistance hash function.
- (KG, S, V) a (t, ε) one-time signature for n-bit documents.

Define a new signature scheme (KG, S', V') as follows:

- Signing $S'_{\mathsf{sk}}(m) := S_{\mathsf{sk}}(H(m))$
- Verifying $V'_{\mathsf{pk}}(m, \sigma) := V_{\mathsf{pk}}(H(m), \sigma)$

Thm: The new scheme is $(t - O(s), 2\varepsilon)$ one-time signature for messages of arbitrary length, where s is the complexity of S, V, H.

Comments

- We did not assume that H is a random function (also known as a random oracle).
- The length of the signature + key is shorter than the length of the document.

Signing Many Messages:
CPA Existentially Unforgeable Stateful Signatures

Q: How to go from one-time security to security under Chosen Plaintext Attack?

Idea: Key refreshing!
Generate new key for each document and certify it using the previous key.

CPA Existentially Unforgeable Stateful Signatures

Let (KG, S, V) be a (t, ε) one-time signature scheme with short keys.

Our new scheme is stateful:

- Generate keys $(\mathsf{sk}_0, \mathsf{pk}_0)$ using KG (key generator).
- To sign the first document m_0:
  generate a pair of keys $(\mathsf{sk}_1, \mathsf{pk}_1)$ and add it to the state,
  output $\mathsf{pk}_1, \sigma_0 = S_{\mathsf{sk}_0}(m_0, \mathsf{pk}_1)$.
- To sign the second document m_1:
  generate a pair of keys $(\mathsf{sk}_2, \mathsf{pk}_2)$ and add it to the state
  output $\mathsf{pk}_1, \sigma_0, \mathsf{pk}_2, \sigma_1 = S_{\mathsf{sk}_1}(m_1, \mathsf{pk}_2)$.
- ...

To verify, follow the "chain of trust" (no not need for remembering a state).

Complexity (time, space, communication) grows linearly with the number of signatures

CPA Existentially Unforgeable Stateful Signatures

Let (KG, S, V) be a (t, ε) one-time signature scheme with short keys.

Our new scheme is stateful:

- Generate keys $(\mathsf{sk}_0, \mathsf{pk}_0)$ using KG (key generation).
- To sign the first document m_0:
  generate a pair of keys $(\mathsf{sk}_1, \mathsf{pk}_1)$ and add it to the state,
  output $\mathsf{pk}_1, \sigma_0 = S_{\mathsf{sk}_0}(m_0, \mathsf{pk}_1)$.
- To sign the second document m_1:
  generate a pair of keys $(\mathsf{sk}_2, \mathsf{pk}_2)$ and add it to the state
  output $\mathsf{pk}_1, \sigma_0, \mathsf{pk}_2, \sigma_1 = S_{\mathsf{sk}_1}(m_1, \mathsf{pk}_2)$.
- ...

CPA Existentially Unforgeable Stateful Signatures

Let (KG, S, V) be a (t, ε) one-time signature scheme with short keys.

Our new scheme is stateful:

- Generate keys $(\mathsf{sk}_0, \mathsf{pk}_0)$ using KG (key generation).
- To sign the first document m_0:
  generate a pair of keys $(\mathsf{sk}_1, \mathsf{pk}_1)$ and add it to the state,
  output $\mathsf{pk}_1, \sigma_0 = S_{\mathsf{sk}_0}(m_0, \mathsf{pk}_1)$.
- To sign the second document m_1:
  generate a pair of keys $(\mathsf{sk}_2, \mathsf{pk}_2)$ and add it to the state
  output $\mathsf{pk}_1, \sigma_0, \mathsf{pk}_2, \sigma_1 = S_{\mathsf{sk}_1}(m_1, \mathsf{pk}_2)$.
- ...

Security?

CPA Existentially Unforgeable Stateful Signatures

Let (KG, S, V) be a (t, ε) one-time signature scheme with short keys.

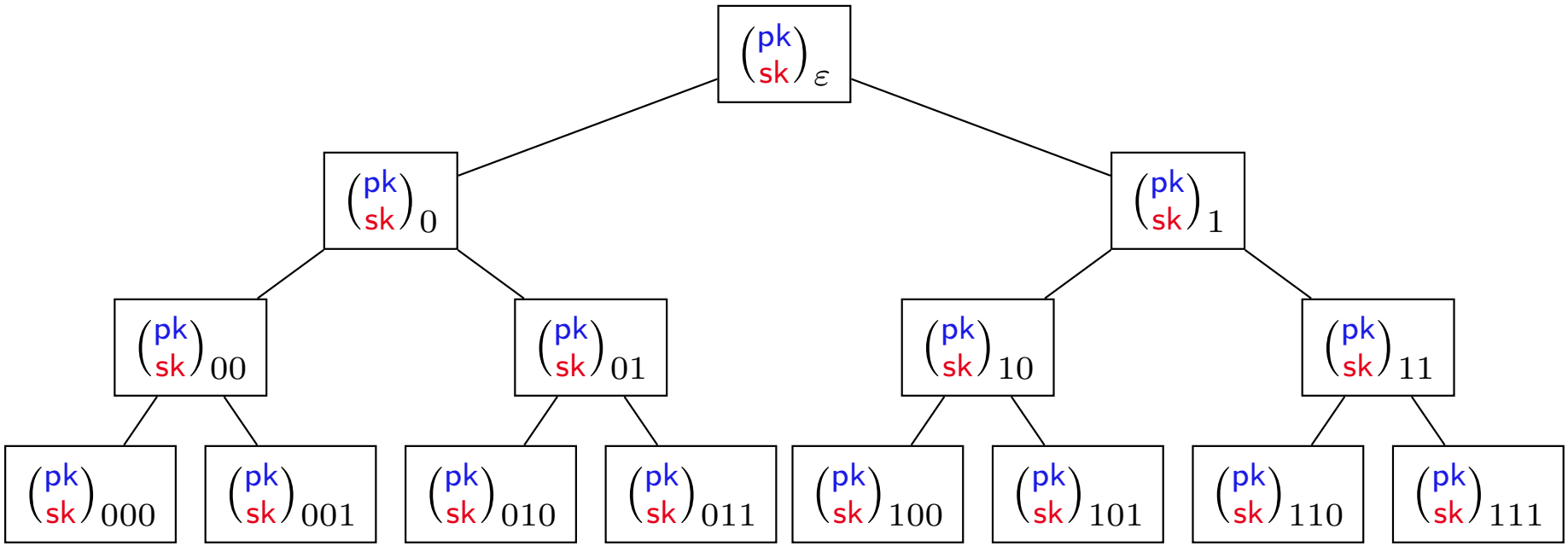
Our new scheme is stateful:

- Generate keys $(\mathsf{sk}_0, \mathsf{pk}_0)$ using KG (key generation).
- To sign the first document m_0:
  generate a pair of keys $(\mathsf{sk}_1, \mathsf{pk}_1)$ and add it to the state,
  output $\mathsf{pk}_1, \sigma_0 = S_{\mathsf{sk}_0}(m_0, \mathsf{pk}_1)$.
- To sign the second document m_1:
  generate a pair of keys $(\mathsf{sk}_2, \mathsf{pk}_2)$ and add it to the state
  output $\mathsf{pk}_1, \sigma_0, \mathsf{pk}_2, \sigma_1 = S_{\mathsf{sk}_1}(m_1, \mathsf{pk}_2)$.
- ...

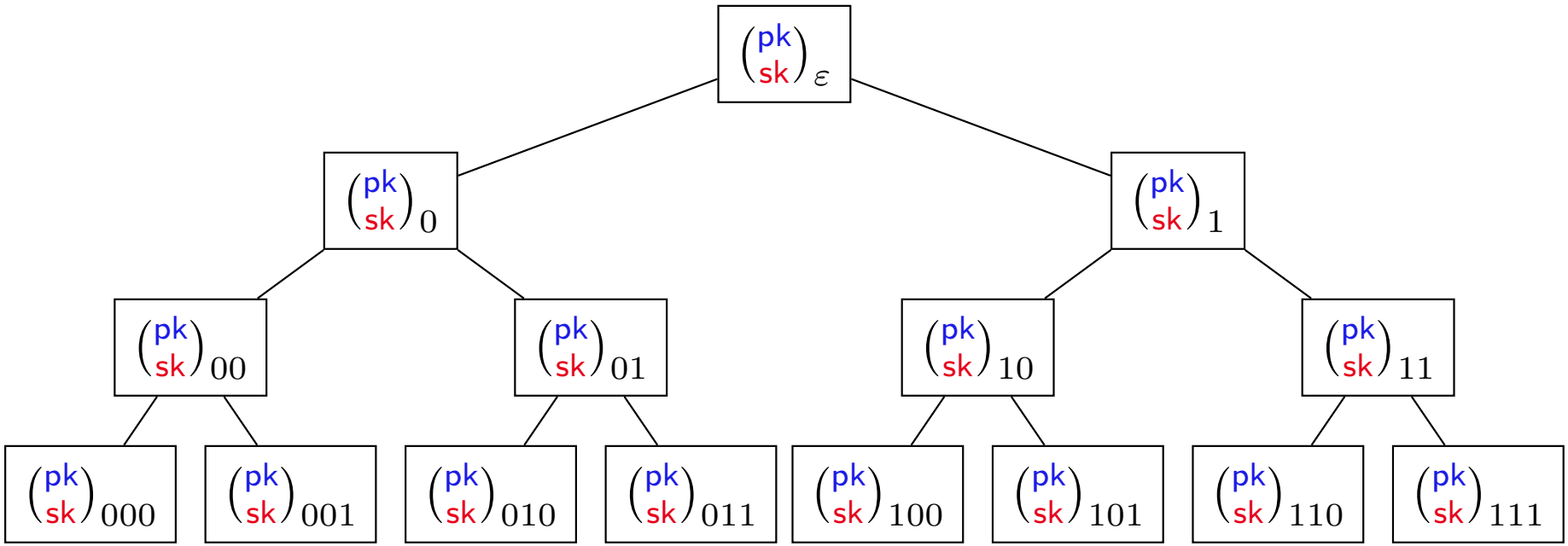
Security? Informally, forging a signature on a new message (chain) implies local forgery under (KG, S, V) (in one node).

Since each sub-key is used once this is infeasible by the security of the one-time signature.

Reducing the complexity of Signing/Verifying

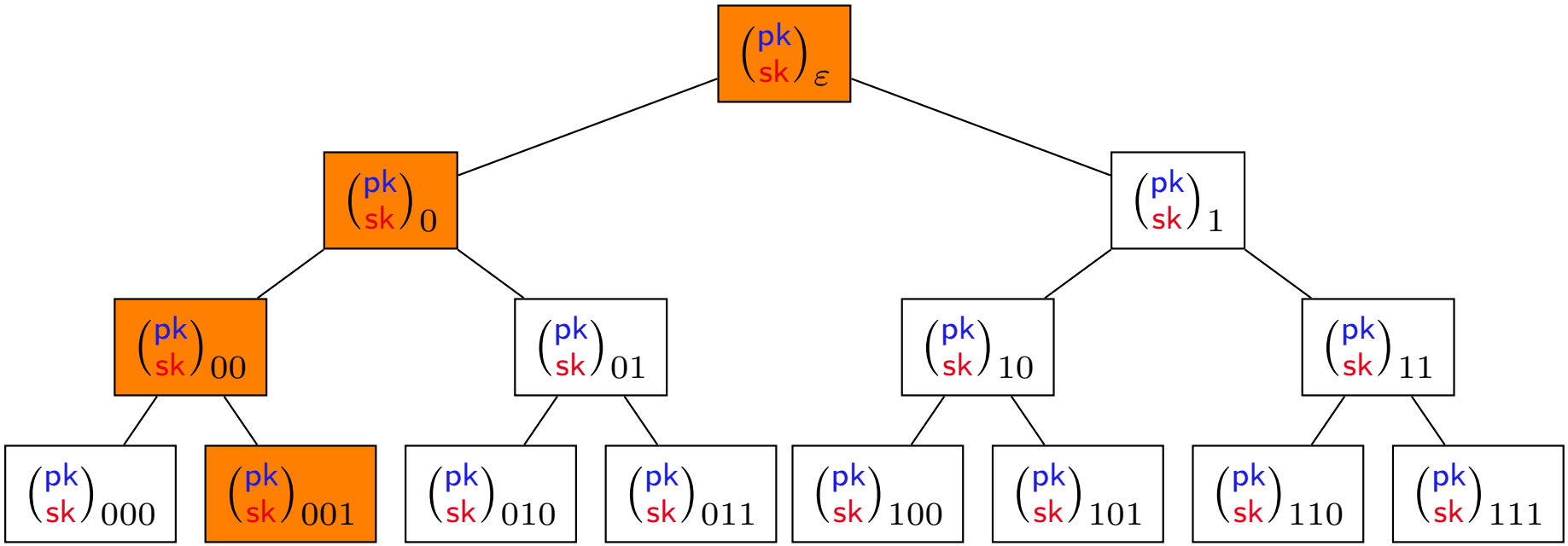


Reducing the complexity of Signing/Verifying



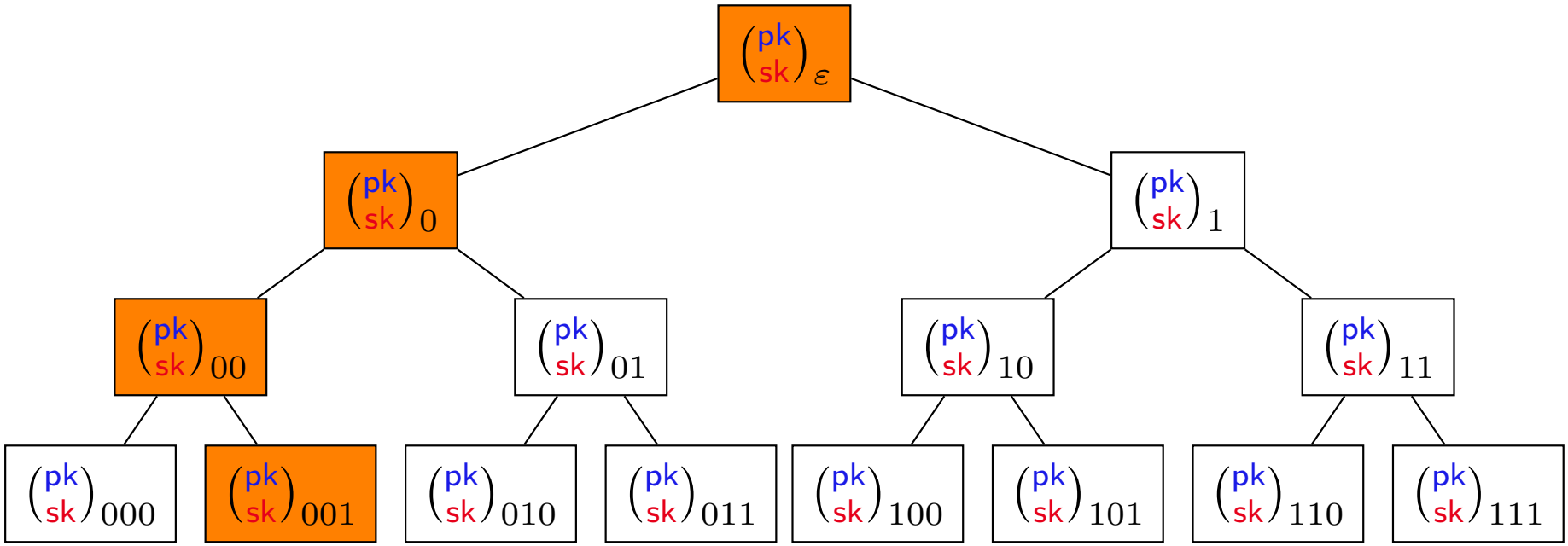
- Full binary tree of depth n
- Nodes at level t are labeled by t-bit strings
- Leaves correspond to messages in $\{0,1\}^n$
- Each node x holds a key-pair $\mathsf{pk}_x, \mathsf{sk}_x$ for one-time signature
- Publish pk_ε as the public-key

Signing



$$
\begin{aligned}
S'_{\vec{\mathsf{sk}}}(001) = & \mathsf{pk}_0, \mathsf{pk}_1, S_{\mathsf{sk}_\varepsilon}(\mathsf{pk}_0, \mathsf{pk}_1) \\
& \mathsf{pk}_{00}, \mathsf{pk}_{01}, S_{\mathsf{sk}_0}(\mathsf{pk}_{00}, \mathsf{pk}_{01}) \\
& \mathsf{pk}_{001}, \mathsf{pk}_{000}, S_{\mathsf{sk}_{00}}(\mathsf{pk}_{000}, \mathsf{pk}_{001}) \\
& S_{\mathsf{sk}_{001}}(001)
\end{aligned}
$$

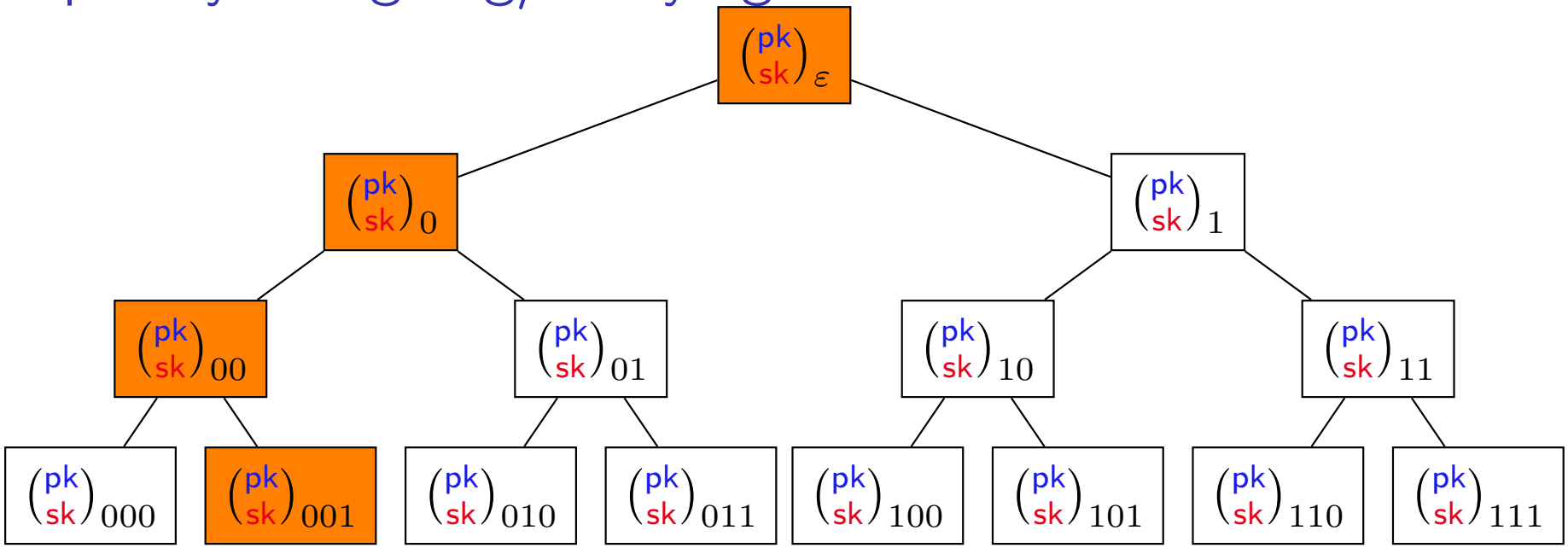
Security



Observations:

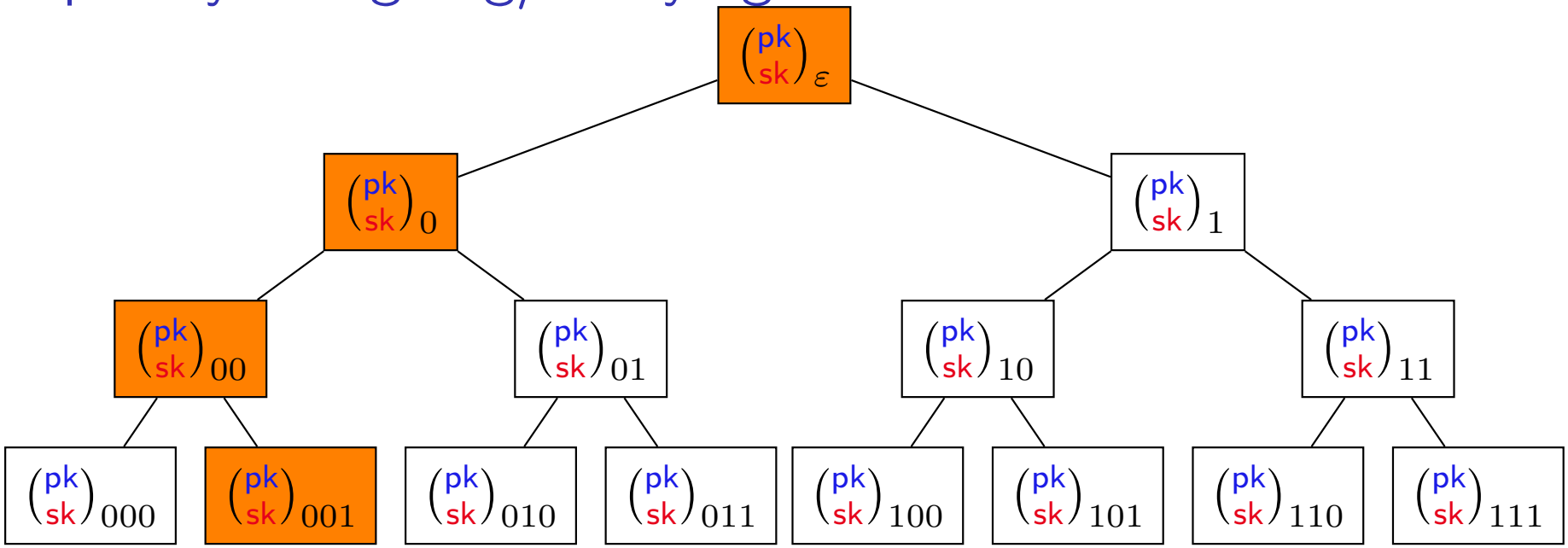
- Each key is used once (for its children)
- new document is always signed by at least one new key

Complexity of signing/verifying



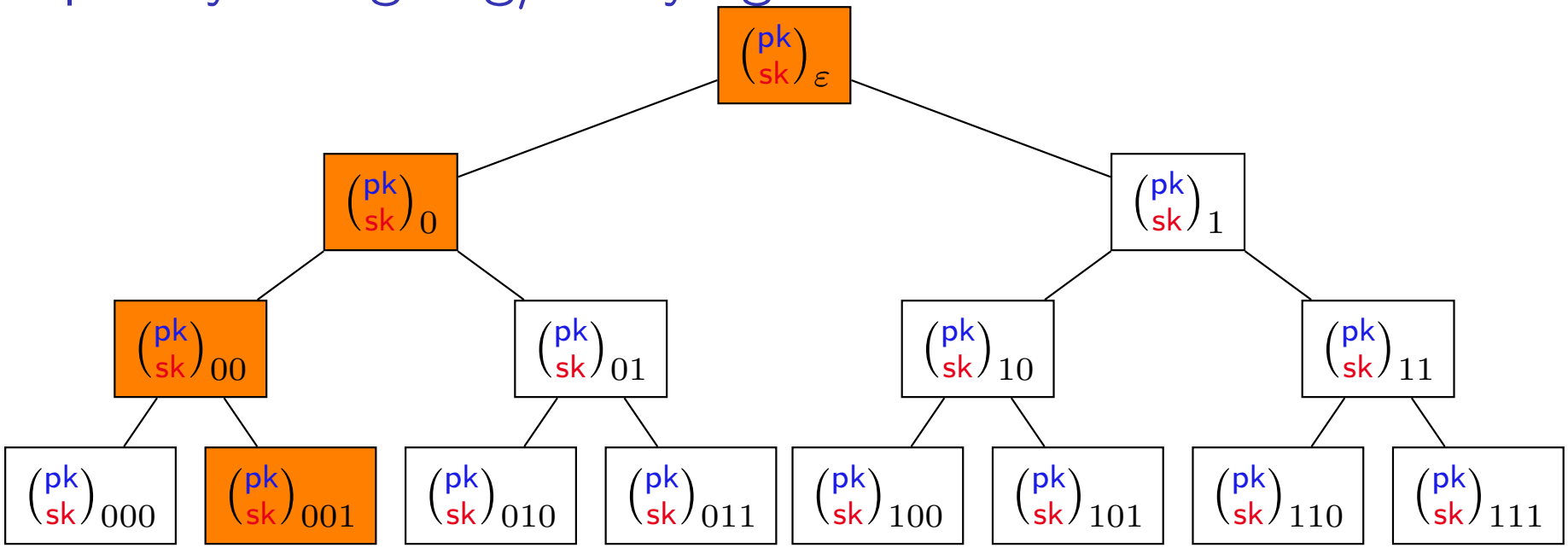
- Signing: walk along the path from root to leaf x, and let each parent (one-time) sign on its children

Complexity of signing/verifying



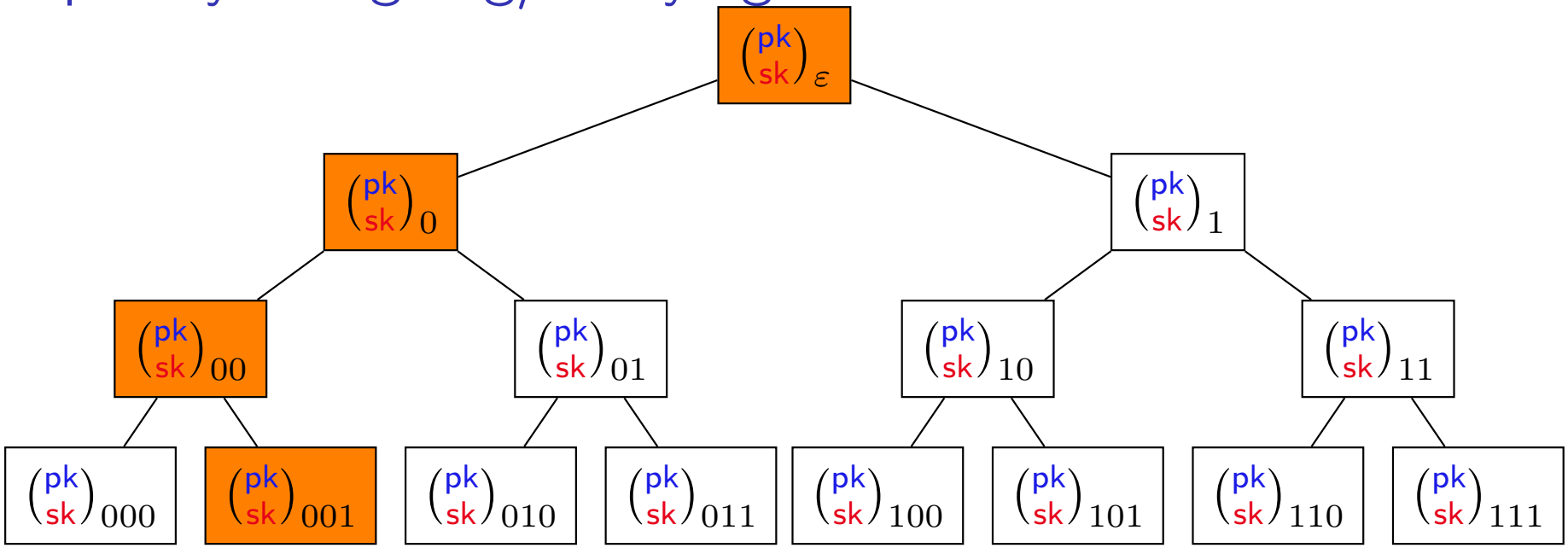
- Signing: walk along the path from root to leaf x, and let each parent (one-time) sign on its children
- Verification is done in the natural way

Complexity of signing/verifying



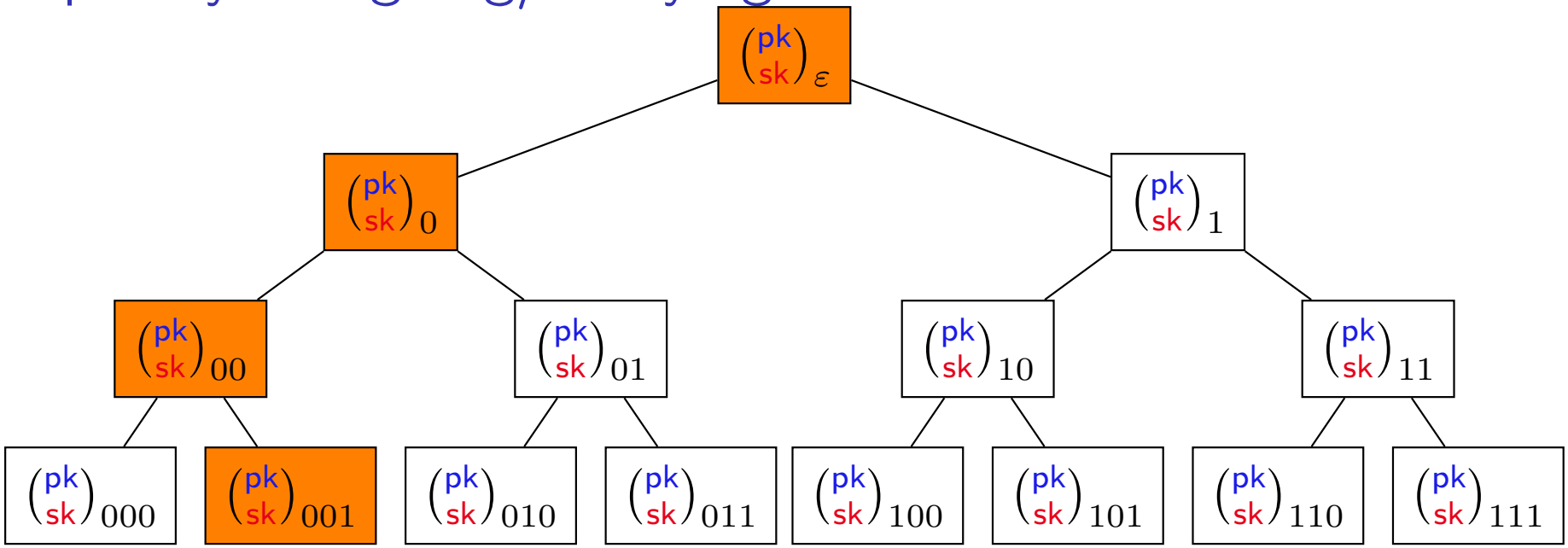
- Signing: walk along the path from root to leaf x, and let each parent (one-time) sign on its children
- Verification is done in the natural way
- Complexity $O(n)$ applications of the one-time signature

Complexity of signing/verifying



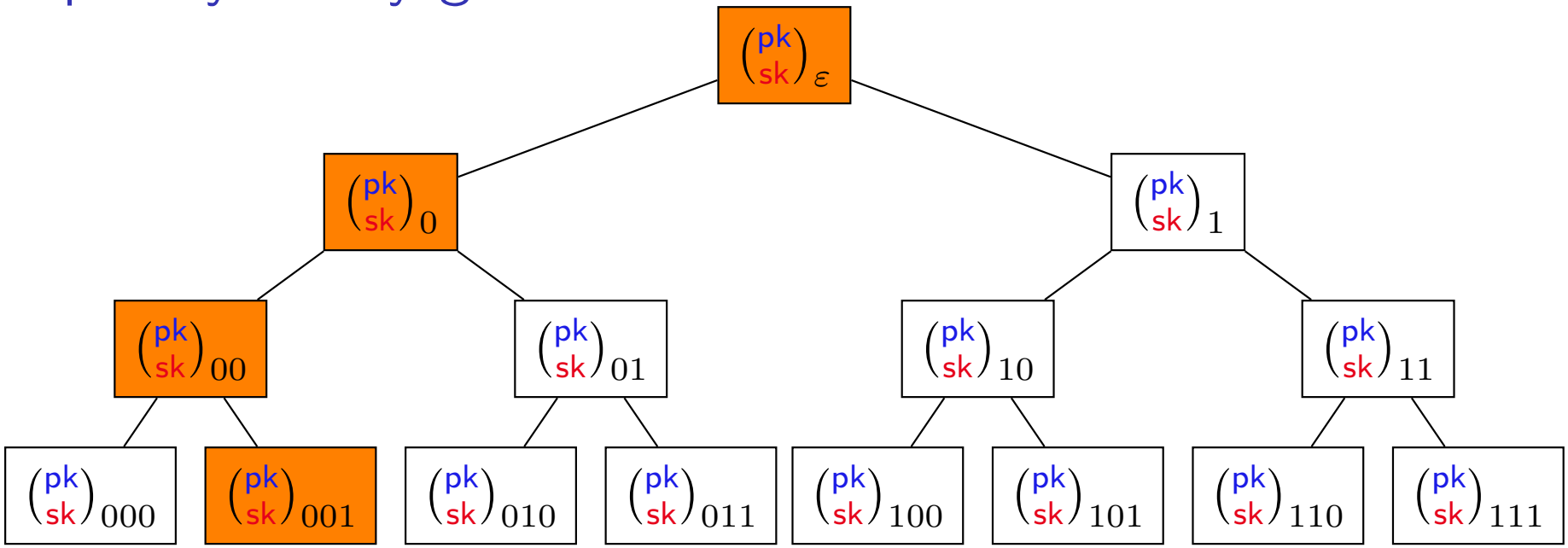
- Signing: walk along the path from root to leaf x, and let each parent (one-time) sign on its children
- Verification is done in the natural way
- Complexity $O(n)$ applications of the one-time signature
- Longer messages can be hashed down to n-bit

Complexity of signing/verifying



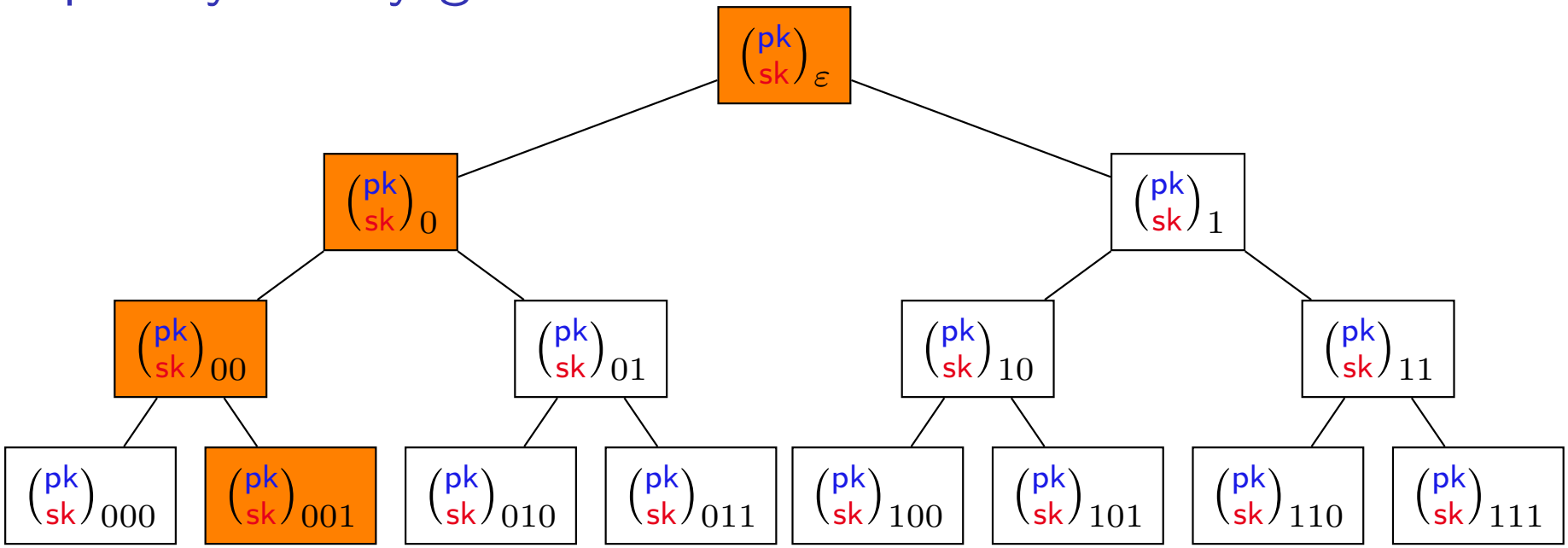
- Signing: walk along the path from root to leaf x, and let each parent (one-time) sign on its children
- Verification is done in the natural way
- Complexity $O(n)$ applications of the one-time signature
- Longer messages can be hashed down to n-bit
- So complexity does not grow with the number of messages :)

Complexity of key-generation



Problem: number of keys across all tree is exponential in n.

Complexity of key-generation



Problem: number of keys across all tree is exponential in n.

- Solution: No need to generate all keys in advance.
- Instead, generate keys on the fly.
- Warning: This must be done in a consistent way! (why?)
- Use a pseudorandom function, F_k and let $(\mathsf{pk}_x, \mathsf{sk}_x) := F_k(x)$ where k is the global secret key.

Take-Home Message

These constructions are quite complicated and require several clever ideas. They do enable us to remove reliance on the random oracle assumption, and use only collision resistant hash functions (an easier to digest assumption) plus one way functions (no trapdoor required), which in turn imply pseudo random functions.

Even though the resulting construction is impractical, it contains several concepts which are widely used in practice

- Hash and Sign
- Key-Refreshing
- Chain of trust
- Use of pseudorandom function to reduce state

Zero Knowledge Proofs[4]

[4] modified from Boaz Barak

Story (Naor, Naor, Reingold)

This is Waldo.

Where is Waldo?

- Daughter: I see Waldo in the picture.

Where is Waldo?

- Daughter: I see Waldo in the picture.
- Dad: I also see him.

Where is Waldo?

- Daughter: I see Waldo in the picture.
- Dad: I also see him.
- Daughter: Really, where?

Where is Waldo?

- Daughter: I see Waldo in the picture.
- Dad: I also see him.
- Daughter: Really, where?
- Dad: Prove to me the you see it, and I'll show him to you

Where is Waldo?

- Daughter: I see Waldo in the picture.
- Dad: I also see him.
- Daughter: Really, where?
- Dad: Prove to me the you see it, and I'll show him to you
- Daughter: ????

Where is Waldo?

- Daughter: I see Waldo in the picture.
- Dad: I also see him.
- Daughter: Really, where?
- Dad: Prove to me the you see it, and I'll show him to you
- Daughter: ????

Where is Waldo?

- Daughter: I see Waldo in the picture.
- Dad: I also see him.
- Daughter: Really, where?
- Dad: Prove to me the you see it, and I'll show him to you
- Daughter: ????

What should the daughter do?

Proofs

- In mathematics and in life, we often want to convince or prove things to others.
- Typically, if I know that X is true, and I want to convince you of that, I try to present all the facts I know and the inferences from that fact that imply that X is true.
- Example: I know that 26781 is not a prime since it is $113 \cdot 237$, to prove to you that fact, I will present these factors, and demonstrate that indeed $113 \cdot 237 = 26781$.

Zero-Knowledge Proofs – Goldwasser, Micali and Rackoff

- Typically, a proof yields some knowledge, beyond the fact that the statement is true.
- In the example, we learned that 26781 is not a prime, and, in addition, we learned its factorization.
- Zero knowledge proof tries to avoid it.

Zero-Knowledge Proofs – Goldwasser, Micali and Rackoff

- Typically, a proof yields some knowledge, beyond the fact that the statement is true.
- In the example, we learned that 26781 is not a prime, and, in addition, we learned its factorization.
- Zero knowledge proof tries to avoid it.

Intuitively:

Zero-Knowledge Proofs (GMR 82')

Alice will prove to Bob that a statement X is true, Bob will be completely convinced that X is true, but will not learn anything as a result of this process.

Zero-Knowledge Proofs – Goldwasser, Micali and Rackoff

- Typically, a proof yields some knowledge, beyond the fact that the statement is true.
- In the example, we learned that 26781 is not a prime, and, in addition, we learned its factorization.
- Zero knowledge proof tries to avoid it.

Intuitively:

Zero-Knowledge Proofs (GMR 82')

Alice will prove to Bob that a statement X is true, Bob will be completely convinced that X is true, but will not learn anything as a result of this process.

One of the most beautiful and influential concepts in CS.
Lead to many applications (*e.g.* practical digital signatures, and hardness of approximation).

Application 1: Identification

- We want to control access to the department.
- Solution: Give authorized people a smart card with a PIN and put a box outside the building that verifies the PIN.
- Problem: Box is outside! Someone may attack it and discover the PIN.
  - by reading the memory.
  - by installing a fake box that records the user's PIN.
- Better if the box contains no secret information at all.

Application 1: Identification

- We want to control access to the department.
- Solution: Give authorized people a smart card with a PIN and put a box outside the building that verifies the PIN.
- Problem: Box is outside! Someone may attack it and discover the PIN.
  - by reading the memory.
  - by installing a fake box that records the user's PIN.
- Better if the box contains no secret information at all.

Solution: Let the box store $f(PIN)$ where f is one-way function. The user proves in ZK to the Box that he knows PIN.

We will see details later.

Application 2: Protocol Design

- Alice & Bob, who don't trust each other, run some crypto protocol.
- "Security" holds if Alice and Bob follow the instructions.
  - e.g., Alice should choose an RSA modulus $n = pq$.
- But what if Alice does not follow the protocol ?
  - e.g., chooses $n = pqr$.
- Security may be lost !
- Bad Solution: Alice sends her inputs and let Bob verify that all is well
  - e.g., reveals n, p, q.
- This is bad for Alice: her inputs are private and she does not trust Bob!

Application 2: Protocol Design

- Alice & Bob, who don't trust each other, run some crypto protocol.
- "Security" holds if Alice and Bob follow the instructions.
  - e.g., Alice should choose an RSA modulus $n = pq$.
- But what if Alice does not follow the protocol ?
  - e.g., chooses $n = pqr$.
- Security may be lost !
- Bad Solution: Alice sends her inputs and let Bob verify that all is well
  - e.g., reveals n, p, q.
- This is bad for Alice: her inputs are private and she does not trust Bob!

Sol: Alice proves to Bob that she followed the instructions of the protocol correctly using Zero-Knowledge Proofs