Secure Function Evaluation vs. Deniability

In OTR and Similar Protocols

bruhns, greg

CONFidence 2012

What and why?

- OTR is a popular cryptographic protocol.
- It features a number of properties, such as confidentiality, integrity and also deniability.
- We're going to focus on the deniability aspect of the protocol.

TOC

OTR

OTR

- Off-the-Record Messaging
- Cryptographic protocol for instant messaging
- Interesting properties
  - Confidentiality
  - Integrity
  - Forward Secrecy
  - Mutual authentication using SMP
  - Deniability

Security Properties

General setting: *Alice* $\leftrightarrow$ *Eve* $\leftrightarrow$ *Bob*

Confidentiality & Integrity

Eve cannot decipher any of Bob's or Alice's messages. Neither can she modify any of those messages (without Alice or Bob noticing that).

Forward Secrecy

Suppose Alice looses her private key. Eve should not be able to decipher any messages that has already been sent in a prior conversation between Alice and Bob.

Mutual authentication

Alice and Bob can authenticate each other.

Security Properties

General setting: *Alice* $\leftrightarrow$ *Eve* $\leftrightarrow$ *Bob*

Confidentiality & Integrity

Eve cannot decipher any of Bob's or Alice's messages. Neither can she modify any of those messages (without Alice or Bob noticing that).

Forward Secrecy

Suppose Alice looses her private key. Eve should not be able to decipher any messages that has already been sent in a prior conversation between Alice and Bob.

Mutual authentication

Alice and Bob can authenticate each other.

Security Properties

General setting: *Alice* $\leftrightarrow$ *Eve* $\leftrightarrow$ *Bob*

Confidentiality & Integrity

Eve cannot decipher any of Bob's or Alice's messages. Neither can she modify any of those messages (without Alice or Bob noticing that).

Forward Secrecy

Suppose Alice looses her private key. Eve should not be able to decipher any messages that has already been sent in a prior conversation between Alice and Bob.

Mutual authentication

Alice and Bob can authenticate each other.

Security Properties

General setting: *Alice* $\leftrightarrow$ *Eve* $\leftrightarrow$ *Bob*

Confidentiality & Integrity

Eve cannot decipher any of Bob's or Alice's messages. Neither can she modify any of those messages (without Alice or Bob noticing that).

Forward Secrecy

Suppose Alice looses her private key. Eve should not be able to decipher any messages that has already been sent in a prior conversation between Alice and Bob.

Mutual authentication

Alice and Bob can authenticate each other.

Security Properties

General setting: *Alice* $\leftrightarrow$ *Eve* $\leftrightarrow$ *Bob*

Deniability

Both, Alice and Bob are hackers and talk about serious stuff (TM) using OTR. Now Alice turns evil and wants to backstab on Bob. Debiability means that Alice can not prove that Bob was really the author of any message he sent (yet, during the conversation Alice is still sure that she talks to Bob).

Key Exchange & Message Crypto

Simplified version of the protocol:

- Each party has an asymmetric key pair, which we'll call master key
- Use Diffie-Hellman to establish a common set of encryption and authentication keys. Alice and Bob sign their Diffie-Hellman messages with their master keys.
  $\longrightarrow$ Authentication and Forward Secrecy
  $\longrightarrow$ Deniability (weak)
- Messages are encrypted (AES-CTR) and MACed (HMAC) using the symmtric keys that have been generated.
  $\longrightarrow$ Confidentiality and integrity

Key Exchange & Message Crypto

- Each message contains a new DH key exchange proposal (also MACed of course). As soon as new keys have been established, the old MAC keys are made public.
  $\longrightarrow$ Deniability (strong)

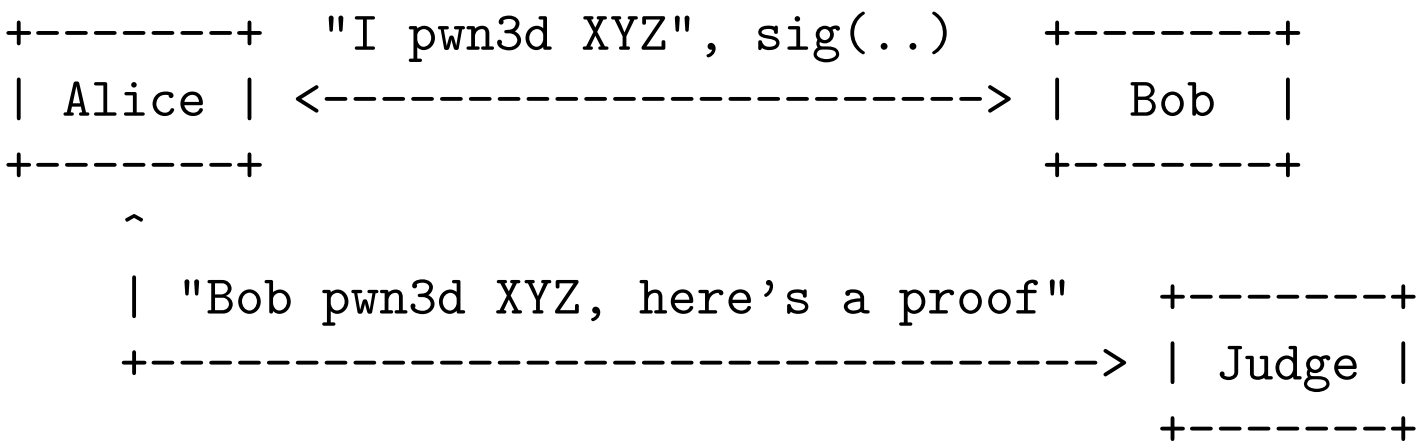
Re-Keying

- Both parties frequently re-key the symmetric primitives.
- The re-keying procedure is significantly simpler than the initial key-exchange - we already have key material exchanged.
- General approach: Perform a DH key-exchange and use already established MAC keys to authenticate the communication.
- Advantage: We can publish the "old" MAC keys, as they are not longer used. We will *not* disclose our encryption keys, though!

Deniability

Deniability

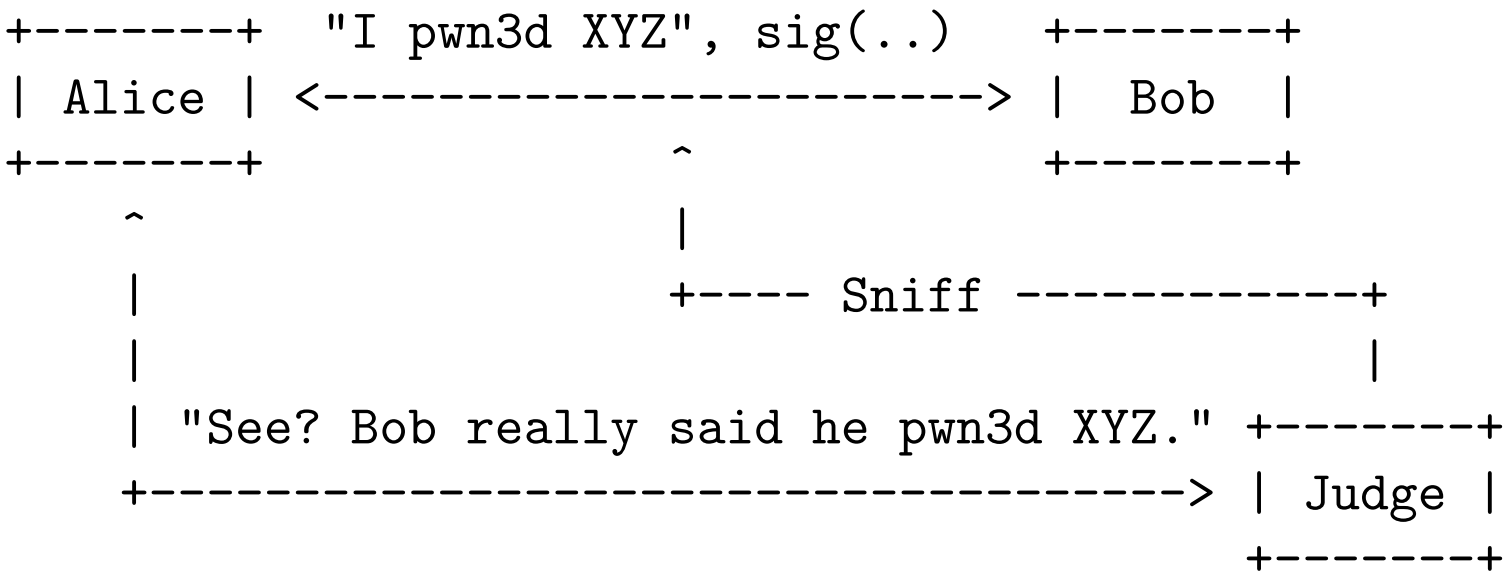
How could Alice attack Bob's deniability?



- Simple offline attack
- Bob will argue that Alice just made up the signature. She can do that, because OTR is deniable (more on that later).
- But Alice might try something else...

Deniability

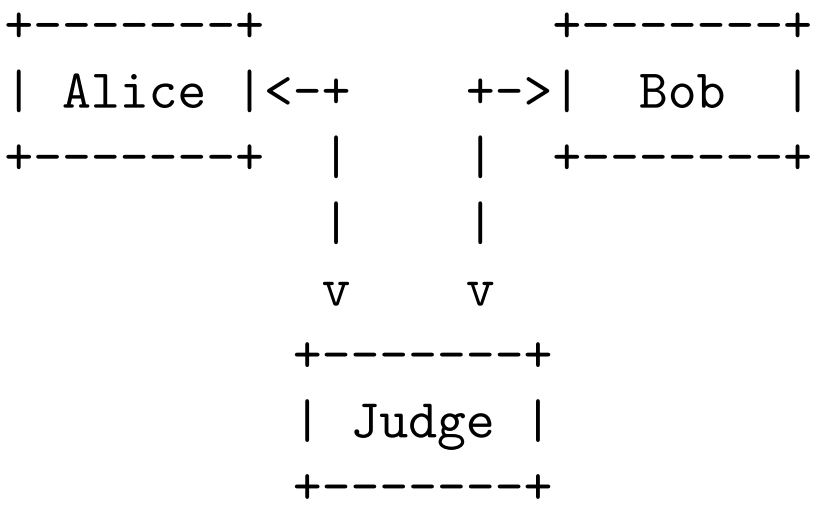
How could Alice attack Bob's deniability?



- If the judge sniffs all traffic, he knows that Bob said. Alice can give him the encryption/MAC key and he can verify that.
- Hard to implement (especially if Alice/Bob use an anonymizer).

Deniability

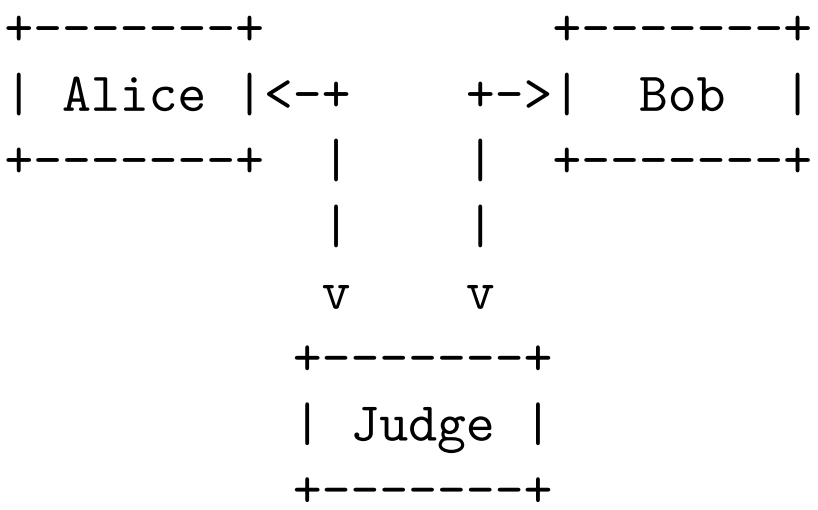
How could Alice attack Bob's deniability?



- Online attack: Alice forwards all traffic to the judge. Also won't work: Bob will again argue that Alice faked the messages.
- Alice gives her master key to the judge, who will act as a proxy for Alice. He will read all messages, check integrity and authentication.

Deniability

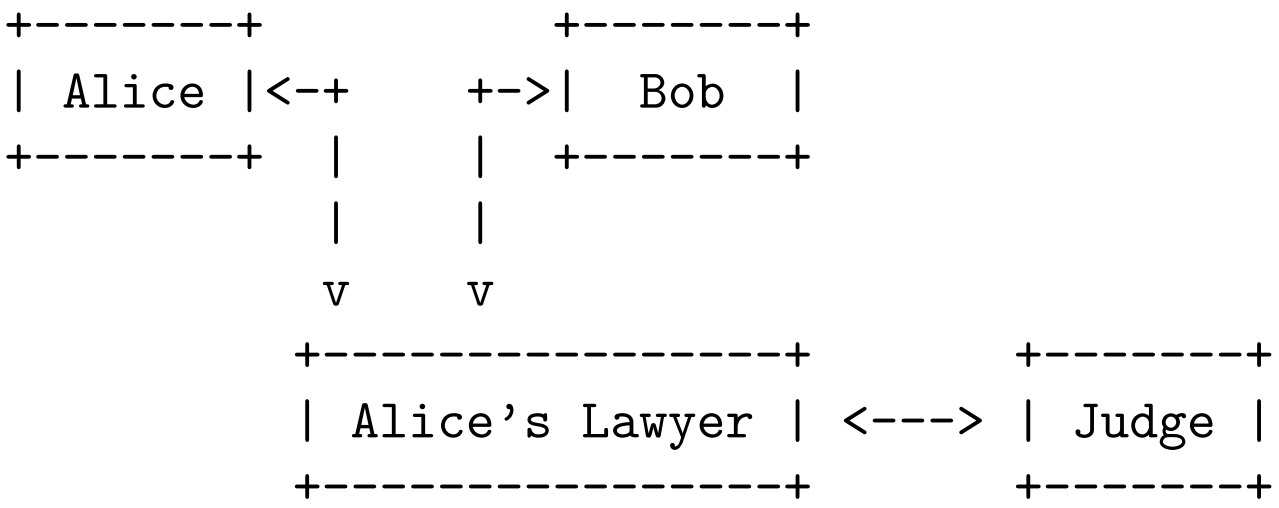
How could Alice attack Bob's deniability?



- But now the judge can impersonate Alice and Alice doesn't really trust the judge.
- What if there was a party that both, Alice and the judge, trust?

Deniability

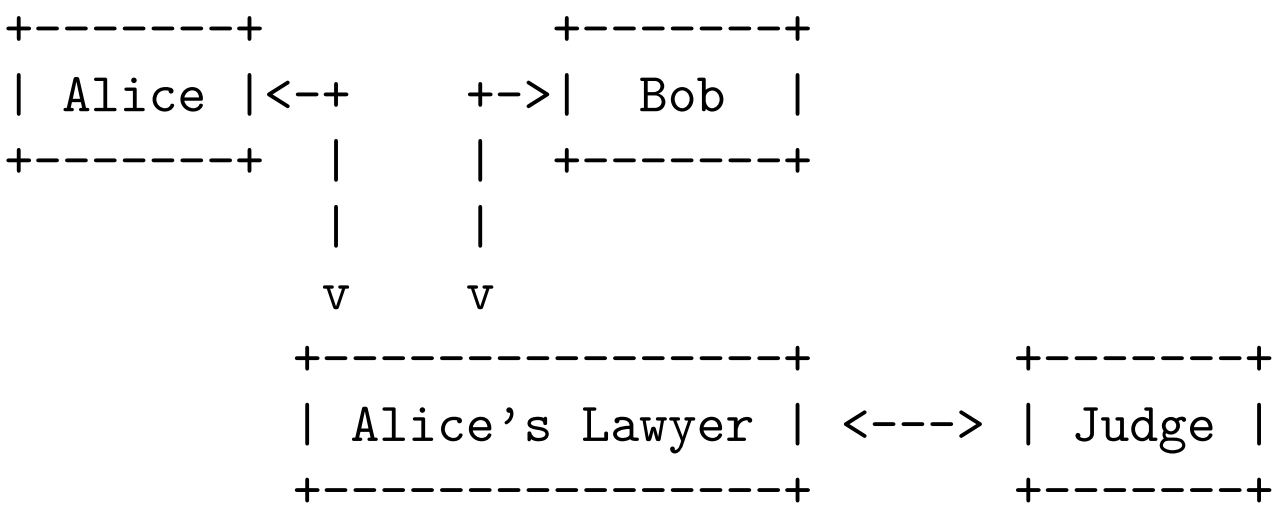
How could Alice attack Bob's deniability?



- Alice just gives her master keys to her lawyer.
- That looks pretty good. But Alice has some doubts...
  - Does the judge really trust her lawyer?
  - Worse: does *she* really trust her lawyer?

Deniability

How could Alice attack Bob's deniability?



- Lucky us: We're all paranoid and there is no trusted third party.
- Alice's goal: get rid of the lawyer!

Deniability

- There are actually two concepts of deniability: weak and strong deniability.
- Weak deniability: If one party discloses a message sent by the other party, the other party can claim that the message was actually faked by the disclosing party.
- Strong deniability: If a message gets disclosed, each party can claim that anybody could have faked that message.
- Weak deniability is pretty obvious: both parties are in possession of encryption and MAC keys, so there is no way to distinguish who actually sent a message.
- Attacks can be offline (observer not involved in the conversation) and online (observer is involved in the conversation).

Strong Deniability

- That's a bit harder. The main idea here is that after each re-keying, the old MAC keys are disclosed.
- In order to fake a message, we need to know two things: MAC key (is public) and encryption key. The encryption key is never disclosed to the public, because that would violate the confidentiality.
- But using the MAC key, we can create a fake re-keying event, so that encryption and MAC keys are generated, which we know.
- Now we can forge any message transcript.

What Does OTR Offer?

- OTR claims strong deniability.
- Deniability does *not* protect against someone who was evil right from the beginning! Such individuals could just share their master keys with any agency.
- If they are able to sniff your traffic, you're screwed, too: they *know*, which messages you sent.
- But sniffing each and every network communication doesn't scale well. Also, if your peer wasn't evil right from the beginning, we cannot assume a trust relation between him and the attacker(s).

Secure Function Evaluation

Secure Function Evaluation

- Remember: Alice wanted to get rid of the lawyer. Doesn't look like an easy task.
- But isn't there any way to "emulate" that trusted third party?
- Crypto teaches us the surprising result: there actually is!
- It's called Secure Function Evaluation (SFE).

SFE

- A method for securely computing $f(x, y)$, where you know x, your peer knows y and neither you nor your peer want the other to learn x or y.
- Different approaches for implementing SFE
  - (Partially) homomorphic cryptosystems
  - Yao's Garbled Circuits

Oblivious Transfer

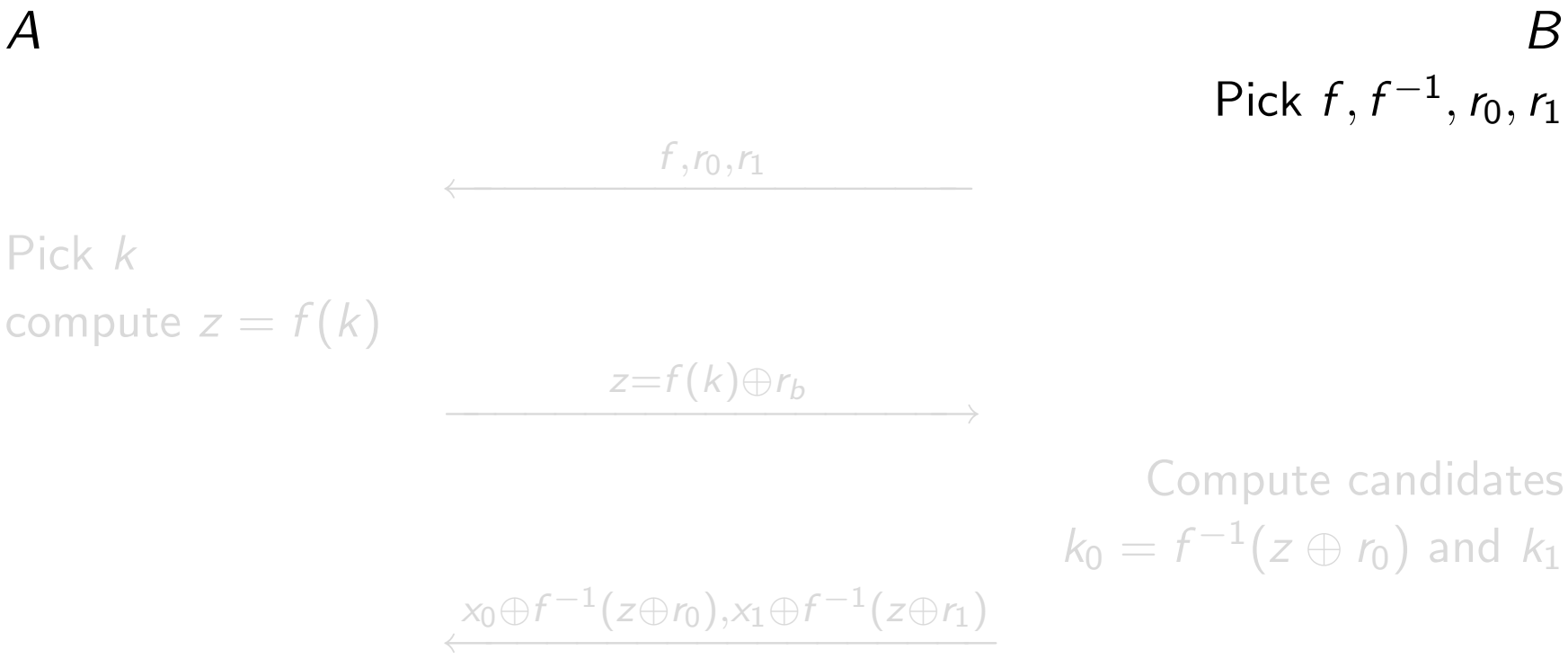
- Bob knows two values, x_0 and x_1. He is willing to share exactly one with Alice.
- Alice wants one of those values, but doesn't want to tell Bob, which one she wants.
- This problem is solved by oblivious transfer.
- There are many OT schemes out there, often based on trapdoor one-way functions. You can imagine those as instances of RSA, where f is the encryption function and f^{-1} the decryption function (keys are fixed).

OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0, 1\}$)

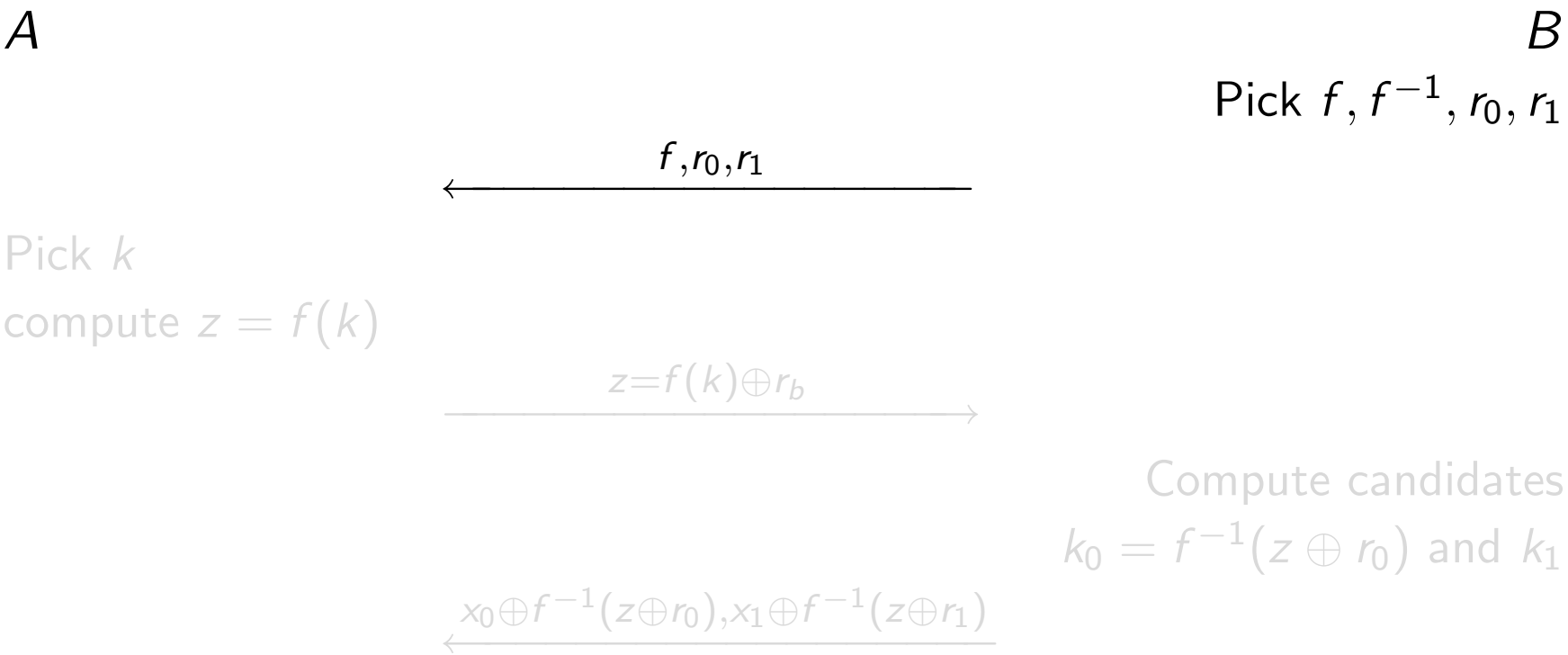
OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0, 1\}$)



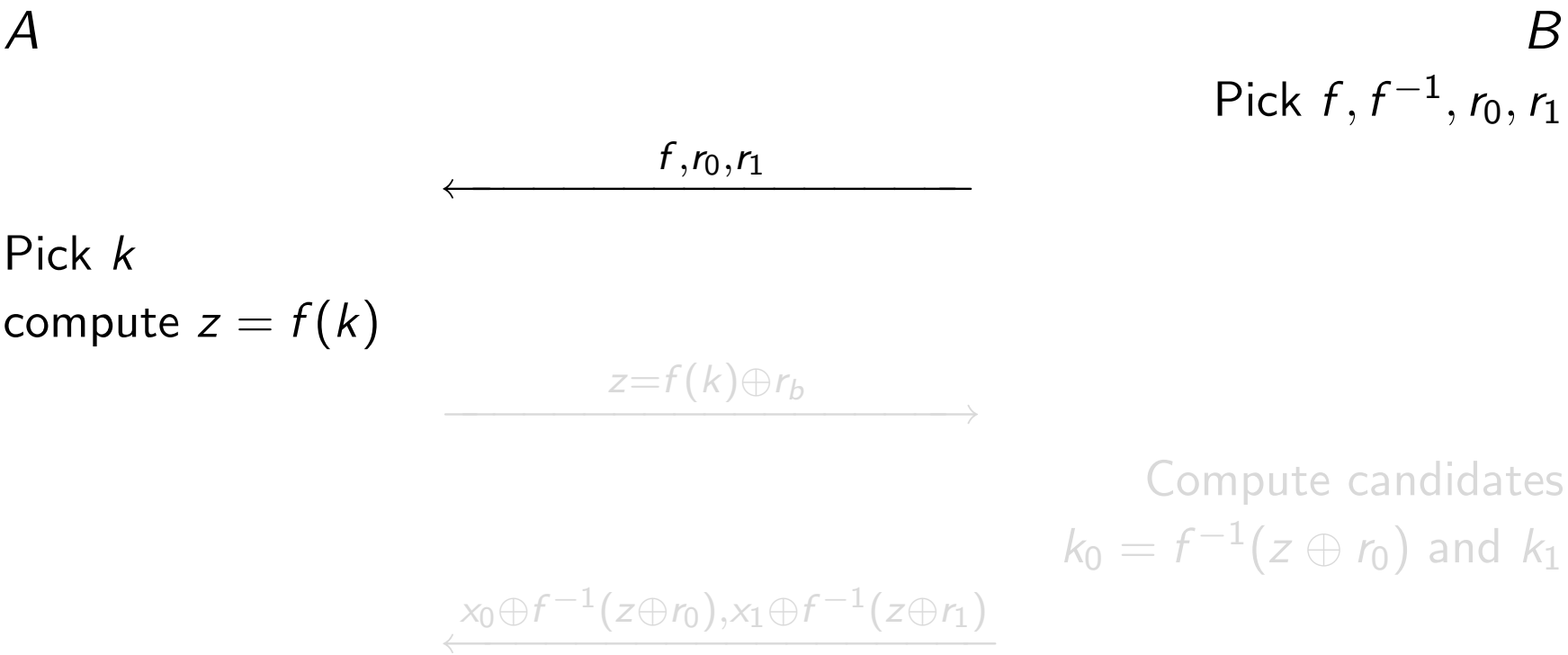
OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0, 1\}$)



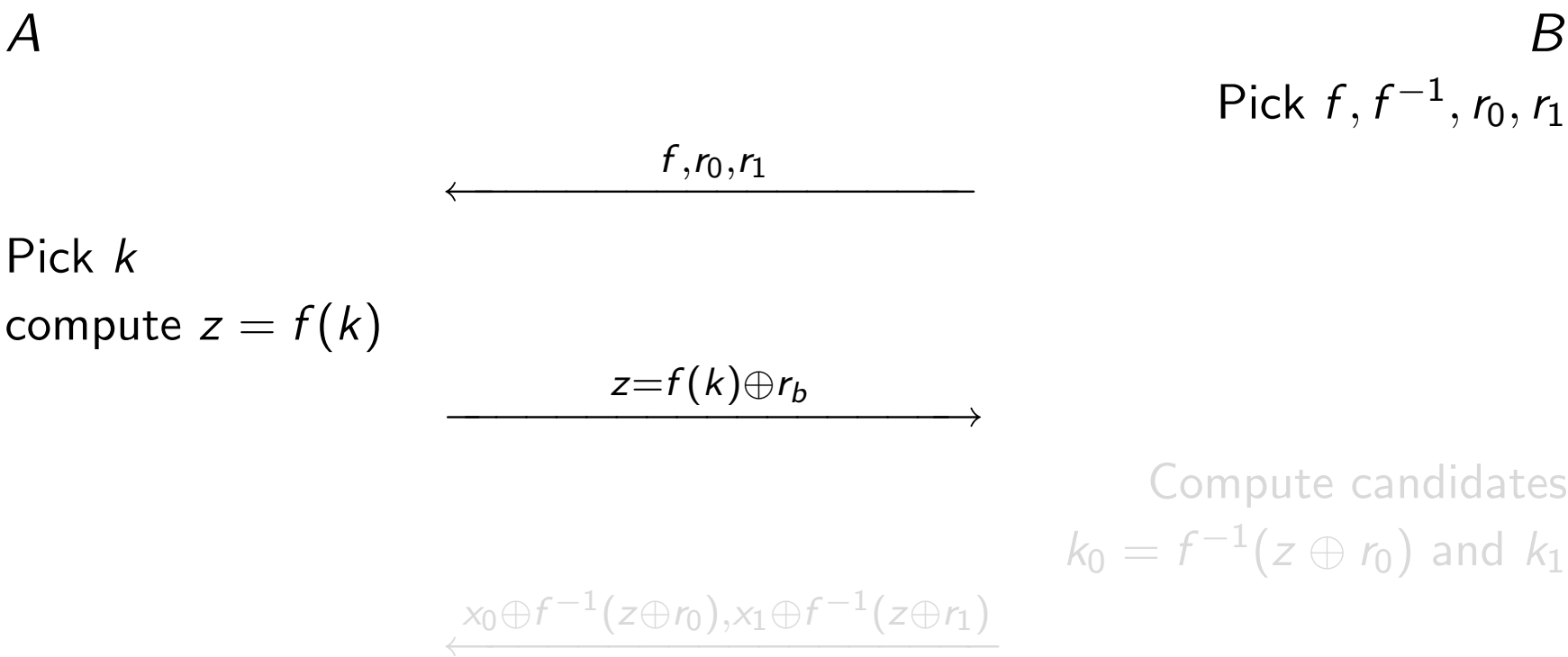
OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0, 1\}$)



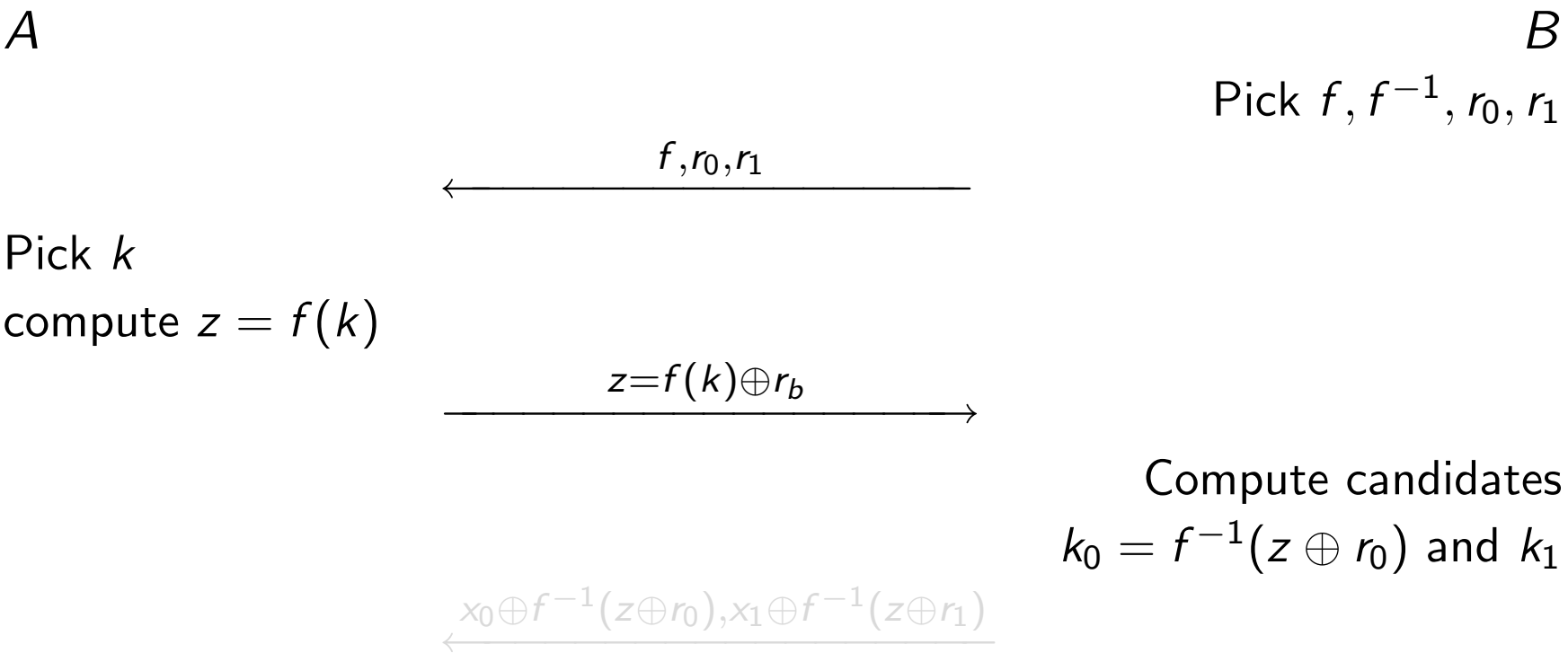
OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0,1\}$)



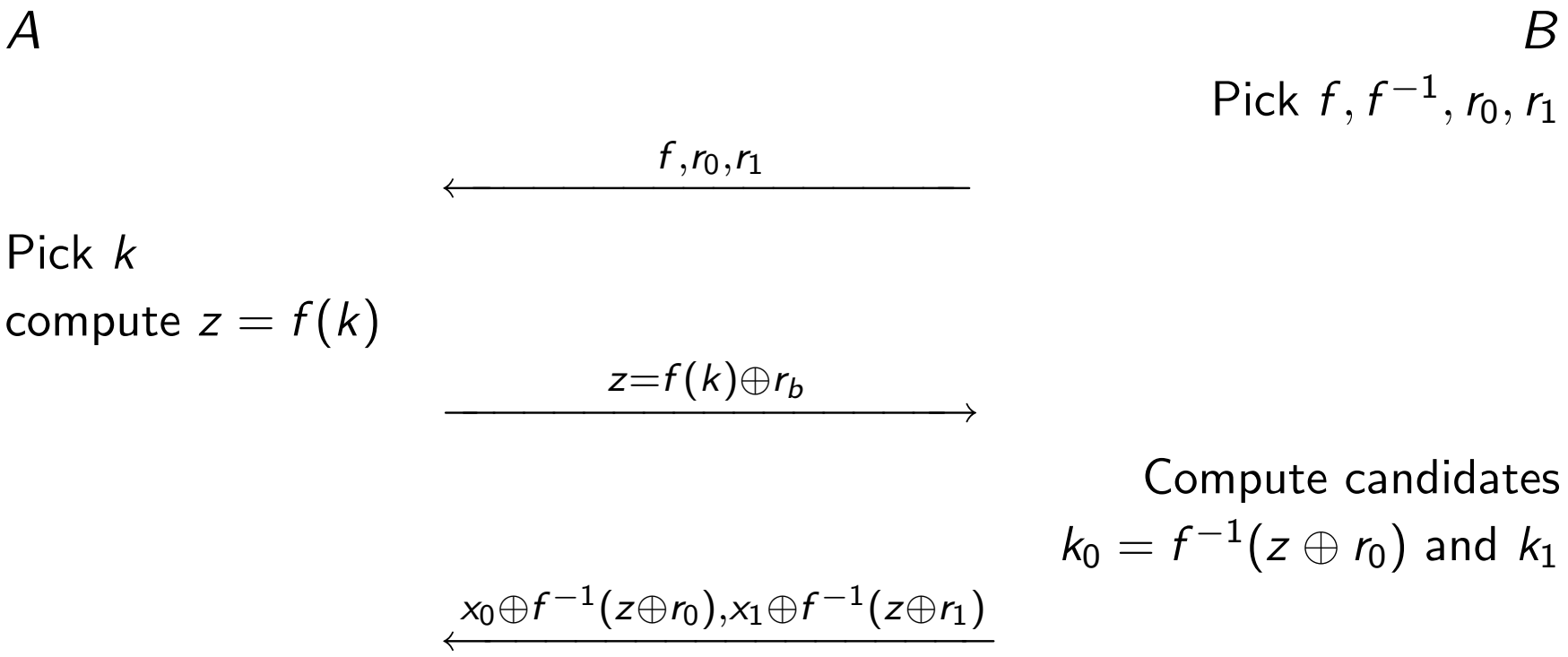
OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0,1\}$)



OT: Compact Version

Bob knows x_0 and x_1, Alice wants to receive one of them (x_b with $b \in \{0, 1\}$)



General OT Approach

- Bob choses f, f^{-1} (trapdoor one-way!) and sends f to Alice. Also, Bob picks two random values r_0 and r_1, which he also sends to Alice.
- Alice wants to retrieve value x_b from Bob. She first generates a random value k and computes $z = f(k) \oplus r_b$, which she sends to Bob.
- Bob computes $k_0 = f^{-1}(z \oplus r_0)$ and $k_1 = f^{-1}(z \oplus r_1)$. He can be sure that one of those k values is correct and the other one is junk. He computes $x'_0 = x_0 \oplus k_0$ and $x'_1 = x_1 \oplus k_1$, which he sends to Alice.
- Alice receives x'_0 and x'_1, but she can only decrypt one of them, because she only knows one k value.

GC SFE

- Evalution of boolean circuits (non-uniform computation)
- Functions can be modeled as circuits.
  - That's what happens when you program an FPGA.
- A circuit consists of gates and wires connecting those gates.
- Alice and Bob agree on a circuit. Alice "garbles" it and Bob evaluates the garbled circuit.
- Gates can be represented by their truth tables. For example:

	0	1
0	0	0
1	0	1

Garbling a Gate

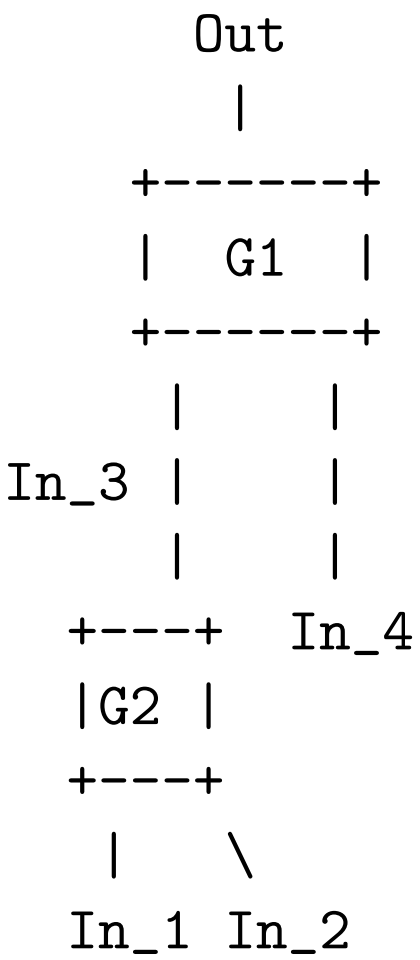
- Main idea: replace the binary inputs by (long) random strings. Instead of 1 or 0, we use keys like 350d5ea01e8a1f407cf or 763581eb6cea7ec4b9a6e.
- Pick such random keys for each value of each wire (if we have inputs a and b, we'll call the keys k_{a0}, k_{a1}, k_{b0} and k_{b1}).
- Encrypt the truth table of the gate using the generated keys. Pick an encryption function E, decryption function D. Let D be built in such a way that it will complain if wrong keys are used (instead of just decrypting junk).

	0	1
0	0	0
1	0	1

$\longrightarrow$

	0	1
0	$E_{k_{a0}}(E_{k_{b0}}(0))$	$E_{k_{a1}}(E_{k_{b0}}(0))$
1	$E_{k_{a0}}(E_{k_{b1}}(0))$	$E_{k_{a1}}(E_{k_{b1}}(1))$

Garbling a Circuit



- When you have the input keys, just try to decrypt all entries in a garbled gate until one operation succeeds.
- In_1, In_2 and In_4 are known (will cover that later)
- But where to get In_3 from? Simple trick: We put the key In_3 into the garbled table of $G2$.

	0	1
0	$E_{k_{a0}}(E_{k_{b0}}(In3_0))$	$E_{k_{a1}}(E_{k_{b0}}(In3_0))$
1	$E_{k_{a0}}(E_{k_{b1}}(In3_0))$	$E_{k_{a1}}(E_{k_{b1}}(In3_1))$

Evaluating a GC

- Where to get the input keys from?
- Alice can hard-code her inputs into the circuit when garbling it.
- Bob however doesn't know the keys for his inputs.
- Bob could just ask Alice to give him the keys for his input values. But either he tells Alice what his input values are or Alice gives him the keys for all input values. Neither is acceptable.
- Solution: Oblivious transfer
- After Bob obliviously received his garbled input values, he evaluates the circuit, sends the result over to Alice and Alice de-garbles it.

Putting It All Together

The Attack Idea

- SFE can be used to emulate a trusted third party.
- That is: Instead of letting a trusted third party compute some function for us, we can just do that ourselves and still have the same security properties.
- Use SFE to emulate a trusted third party: Somehow (TM) share our keys with that "third party".

Sharing is Caring

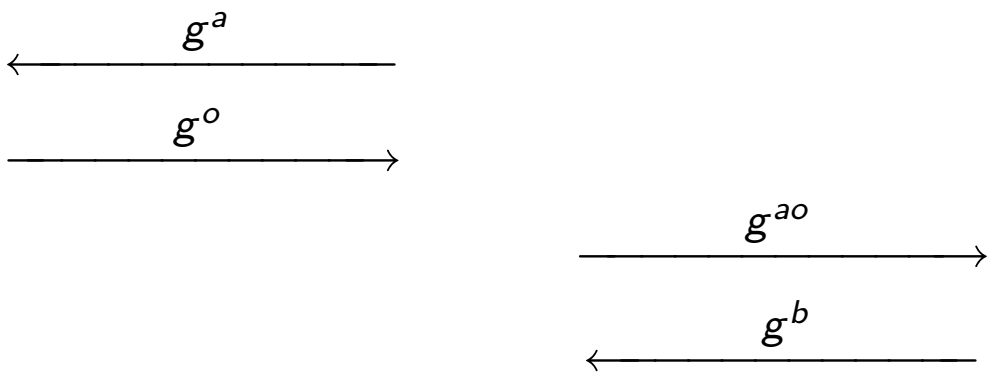
- We want to backstab our peer, therefore we collaborate with an observing party O (a.k.a. the judge or simply *them*).
- We keep our asymmetric master keys for ourselves.
- But every DH key exchange, we compute together with O. The resulting MAC keys are shared between O and us (i.e. neither O nor we know the keys, details later).
- The encryption keys are only known to us (otherwise, O could learn the plain text).
- To verify a message's integrity, we collaborate with O. For signing a message, we also collaborate with O.

Recap: Diffie-Hellman

- A and B publicly agree on a prime p and a generator g of a large cyclic subgroup of $\mathbb{Z}_p$.
- A picks a random a, computes g^a. Same for B.
- A and B exchange g^a and g^b, but keep a and b secret.
- A computes $k = (g^b)^a$, B computes $k = (g^a)^b$. Both keys are the same $k = g^{ab} = g^{ba}$.
- An attacker cannot compute g^{ab} from g^a and g^b alone (Diffie-Hellman problem).

The Attack Implementation

- Cooperative key-exchange: consider three parties A, B and O (Alice, Bob and the observing party). A and O cooperate.



- B now knows g^{abo}, but neither A not O can compute that value.
- Proof idea: A knows g^b, g^o and a. Computing $g^{oab} \leftrightarrow$ computing g^{bo}, equivalent to the Diffie-Hellman problem.

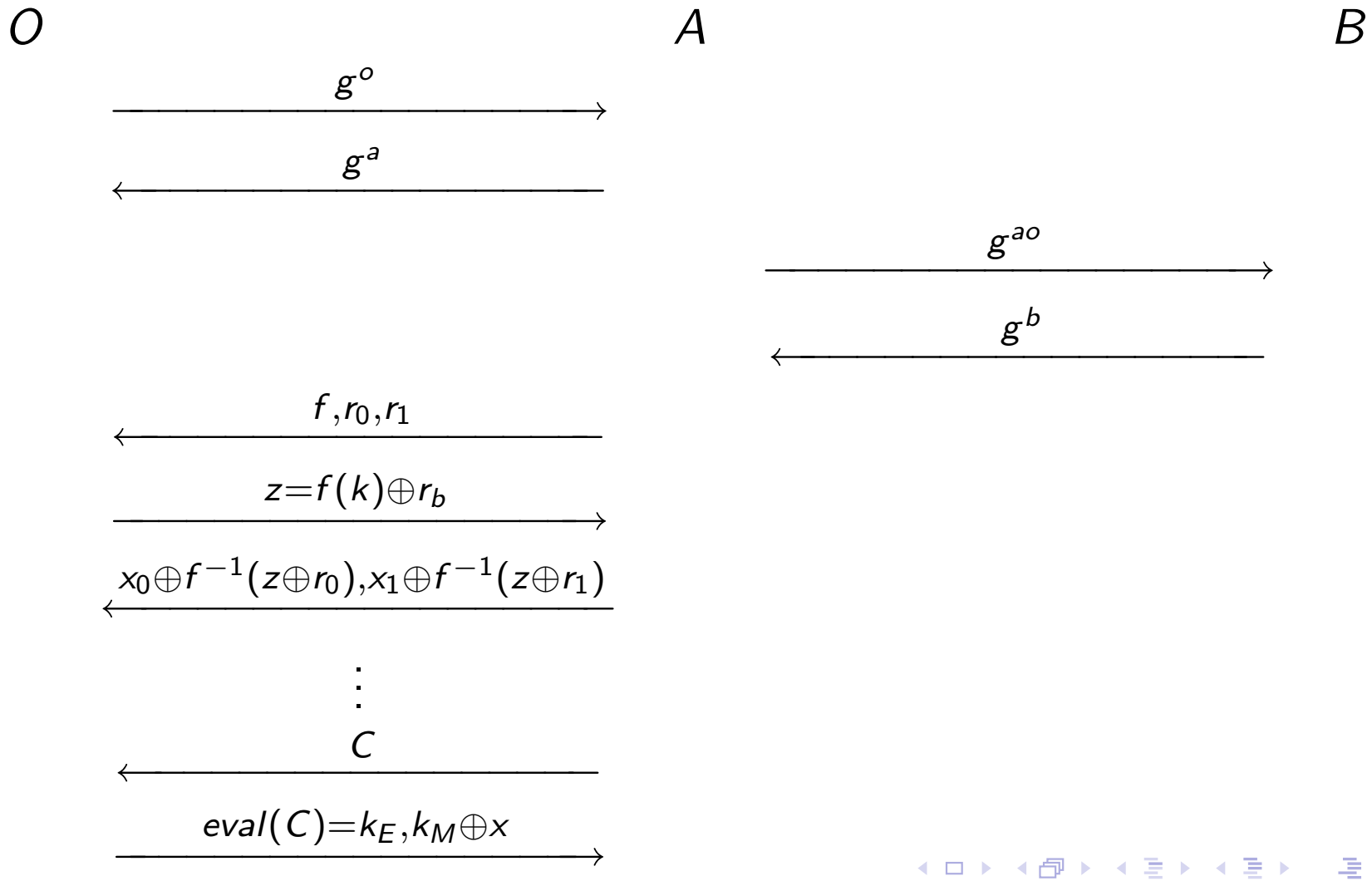
The Attack Implementation

- No SFE up to here!
- A and O both know g^b. With their private exponents, A and O can compute g^{abo}. But A and O don't want to share a and o.
- From g^{abo}, A and B can compute the encryption and MAC keys (those are generated by hasing g^{abo} in various ways).
- The function that A and O want to compute is $f(a, b) = (k_E, k_M) = (HASH_E(((g^b)^a)^o), HASH_M(((g^b)^a)^o))$.
- Inputs of the function are the private exponents a and o.
- The circuit for that is rather big...

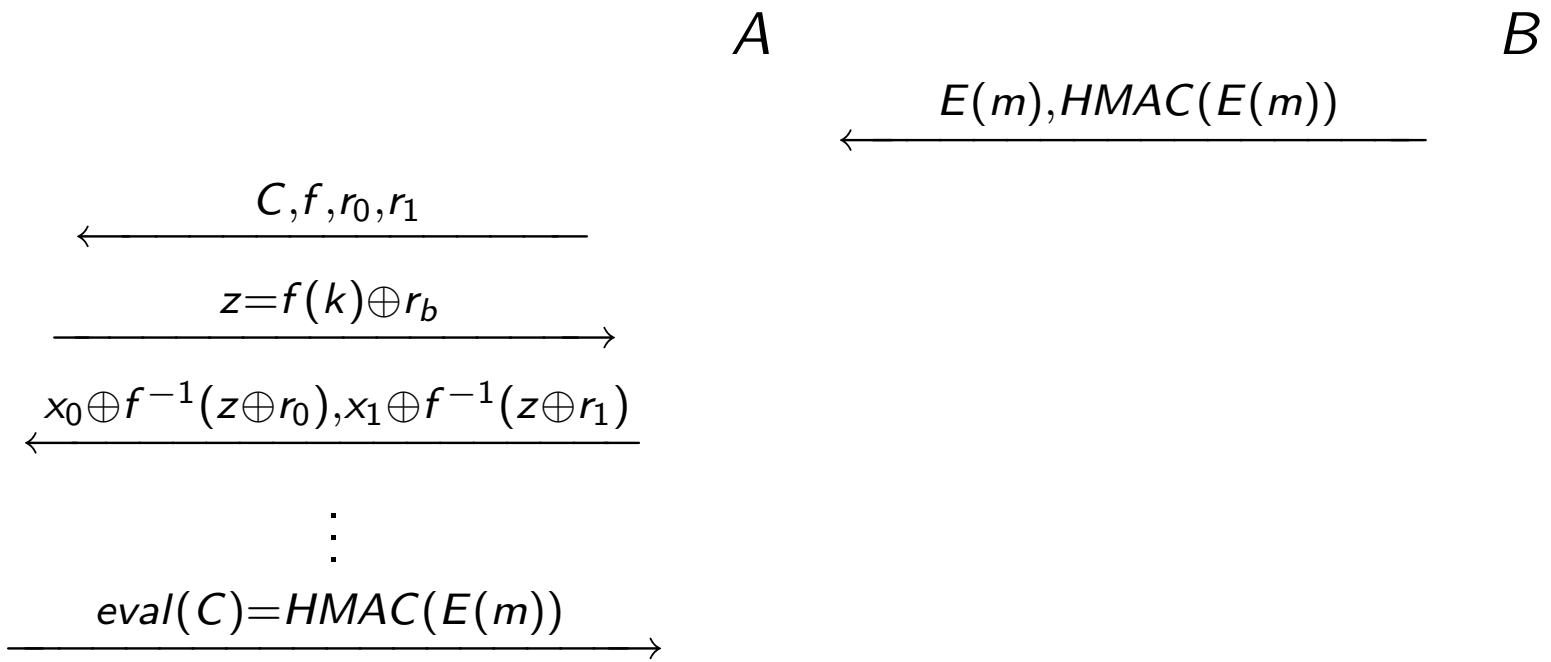
Cleanup

- But now O knows the MAC key and could impersonate A and B...
- Fix: A has to get k_E and A, O have to share k_M!
  - Easily possible by using a circuit that blinds some of its outputs with a key only known to one party: The circuit takes an additional input x and instead of k_M it really computes $k_M \oplus x$.
  - A will know $k_x := k_M \oplus x$ and O will know x.
- For every MAC computation, A and O will jointly compute $f(k_x, x, m) = HMAC(m, k_x \oplus x)$.

The Big Picture: Key Exchange



The Big Picture: Message Exchange



Security Properties of Our Scheme

- O will never see any plain text, so confidentiality stays.
- Without our help, O cannot sign or verify any message, so integrity also stays intact.
- Forward secrecy also remains OK, because O won't learn any decryption keys anyway.

Security Properties of Our Scheme

- But deniability is gone: As we need the help of O to verify the messages of our peer, O learns about the integrity of the messages at the same time as we do.
- Messages that are created afterwards will be rejected by O, just because O has not seen those during the conversation.
- Even worse: We can selectively disclose any message content to O, just by telling O the decryption key for that particular message!
- **This scheme can be extended to other protocols as well!**

What's the Benefit of Our Scenario?

- No trust relationship with O is required (on no side).
- O doesn't have to sniff anything.
- You don't have to disclose everything you or your peer said.
- No "evil" intentions required.
- People might use it as a defense: "just to be sure.. if my peer turns evil at some point in time then I'll disclose everything he said!"

Closing remarks

- "So who is that obverving party?"
  - They might be the NSA or anybody else.
  - Probably the adversary you're most afraid of.
- "Yeah, but you know what: they still cannot prove anything to (yet) another party. And in my country, you actually need to prove stuff in court!!1"
  - Might be (are you sure?)
  - But what if they are the judges / the jury?
- Cooperating "just to be safe" $\longrightarrow$ prisoners dilemma

Implementation Considerations

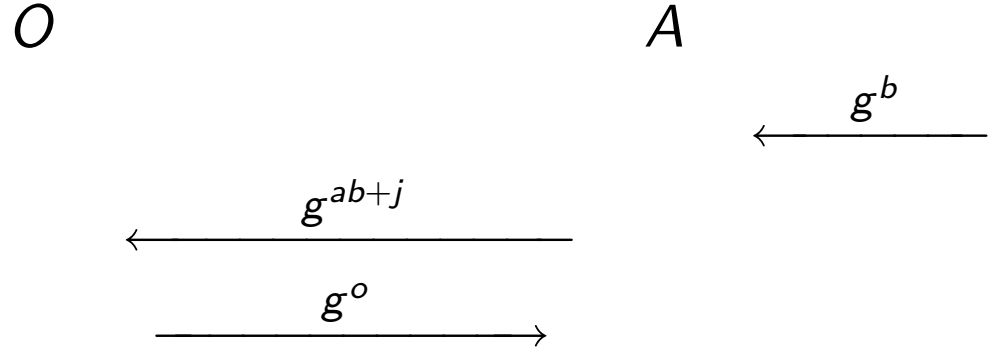
- A real-world implementation would be more complicated (OTR has some restrictions that we left out) but can be done.
- You need some SFE framework (try TASTY and/or Fairplay).
- SFE comes with some performance impact, so think about
- additional optimizations.
- Also, some SFE schemes are by default only secure against honest-but-curious attackers. Need to add zero-knowledge or other tricks to improve that.
- For additional details, please check out our article in Phrack #68.

OTR Related Details

- OTR mandates $k_M = Hash(k_E)$, so A is not allowed to know k_E. Can be solved by also sharing k_E with O and cooperating for encryption and decryption.
- A needs to convince O that she doesn't do re-keyings that O doesn't know about (can be done by using zero-knowledge proofs).
- OTR mandates you should publish your MAC keys when you won't use them anymore. The current implementation doesn't check that but if it would, we'd also need to do that cooperatively.

Optimizing DH

- Computing modular exponentiation is expensive. Even more so in a circuit. Fortunately, DH can be tweaked, so that we only need to do a multiplication in SFE.
- A picks some random value j and does the following:



- O computes $(g^{ab+j})^o = g^{abo+jo}$
- A computes g^{-jo} (trivial to invert g^j)
- Using SFE, A and O compute $g^{abo} = g^{abo+jo} \cdot g^{-jo}$

Questions?

Get in touch:

Mail: ping@gregorkopf.de, twitter: teh_gerg
Mail: bbrehm@math.fu-berlin.de, twitter: bruhn5