



Kryptographie

Prof. Dr. Björn Grohmann

YAO'S MILLIONAIRES PROBLEM



Algorithm 1 Comparing protocol constructed by logical gates

Input: $a = (a_n \cdots a_2 a_1)_2$, $b = (b_n \cdots b_2 b_1)_2$

$f_1 = \bar{a}_1 \wedge b_1$

for $i = 2$ to n **do**

$f_i = \bar{a}_i \wedge b_i \vee (\bar{a}_i \vee b_i) \wedge f_{i-1}$;

end for

Output: f_n

if $f_n = 0$ then $a \geq b$

if $f_n = 1$ then $a < b$

GARBLED CIRCUITS



$b \backslash a$	0	1
0	0	0
1	0	1

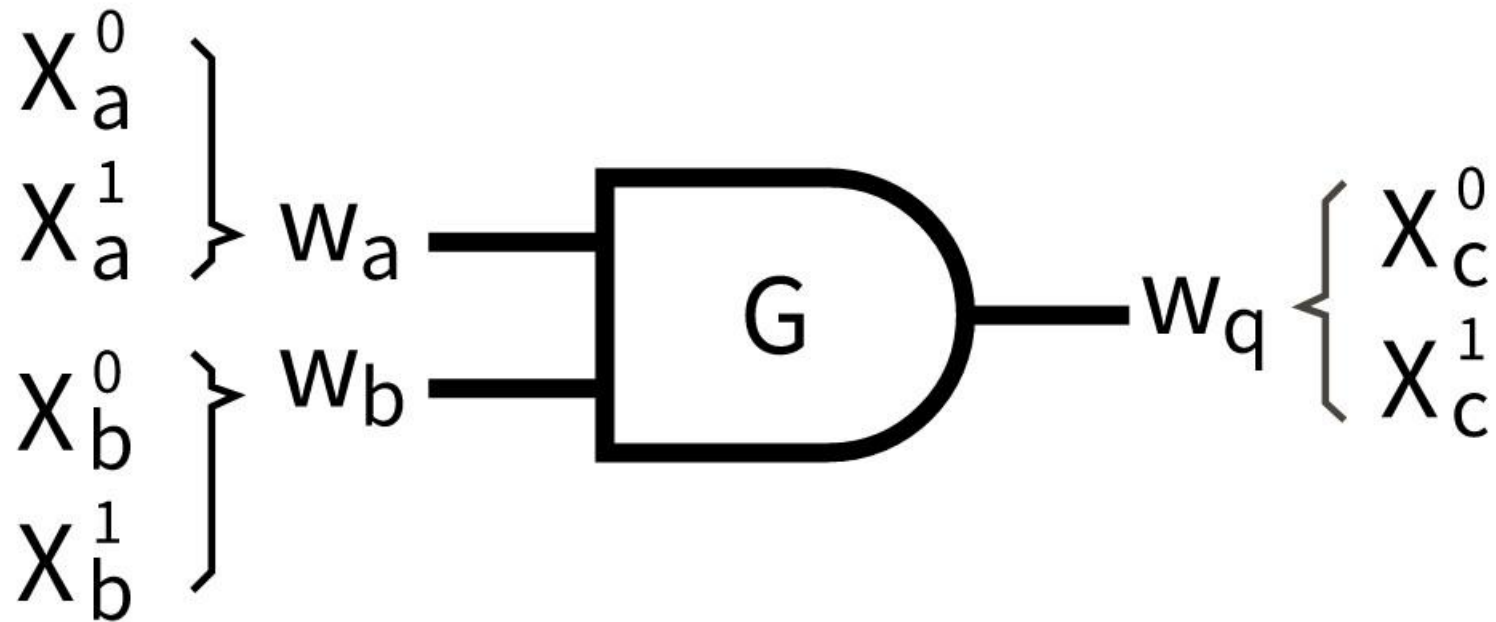


$b \backslash a$	X_a^0	X_a^1
X_b^0	X_c^0	X_c^0
X_b^1	X_c^0	X_c^1

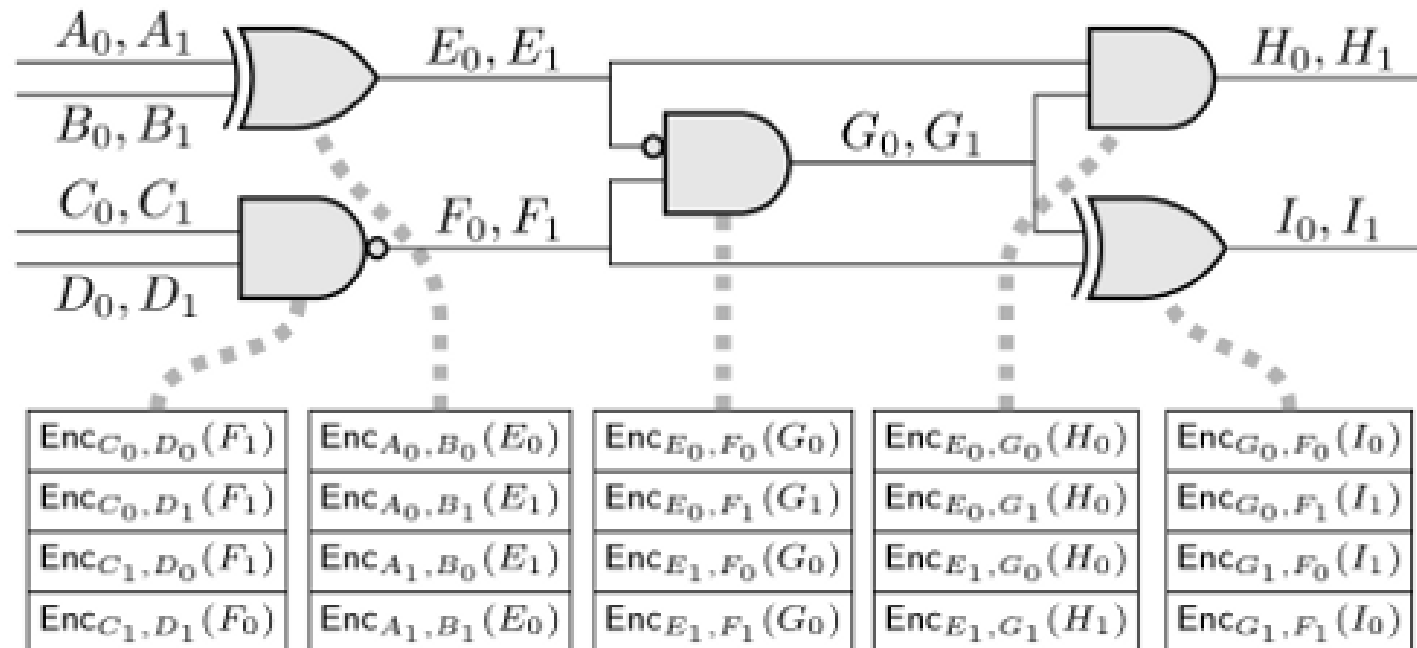


$b \backslash a$	X_a^0	X_a^1
X_b^0	$E_{X_a^0, X_b^0}(X_c^0)$	$E_{X_a^1, X_b^0}(X_c^0)$
X_b^1	$E_{X_a^0, X_b^1}(X_c^0)$	$E_{X_a^1, X_b^1}(X_c^1)$

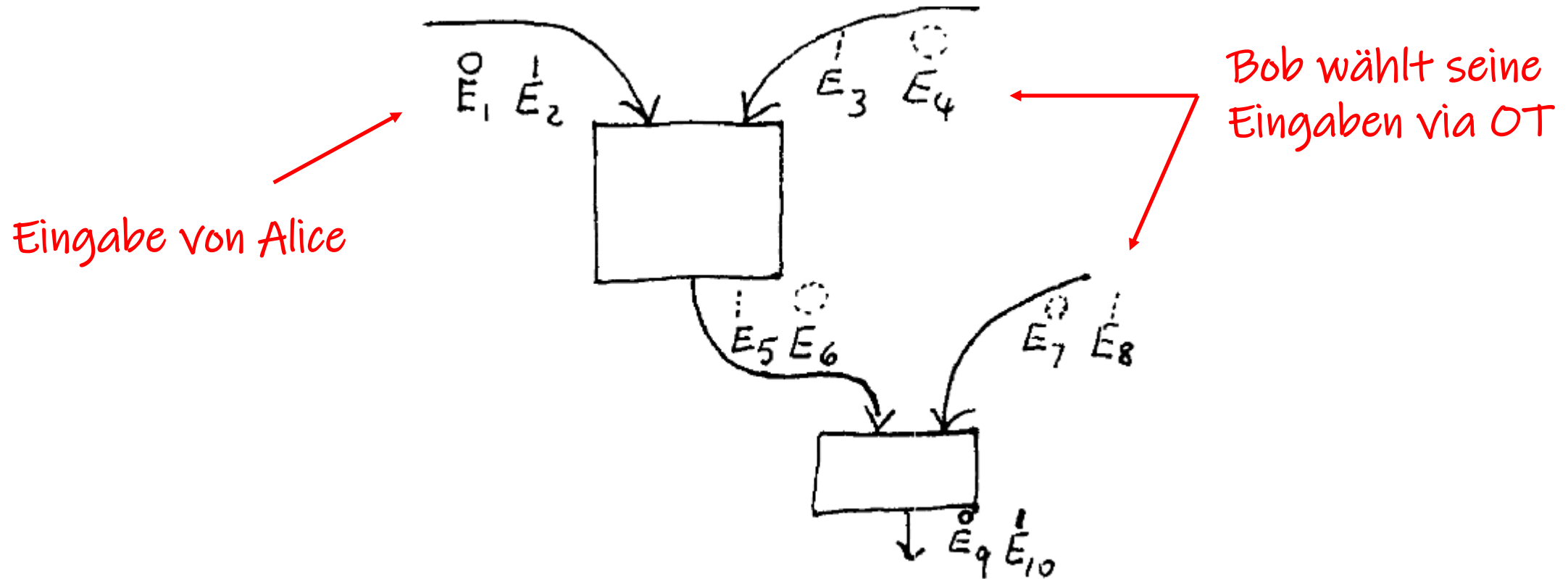
GARBLED CIRCUITS



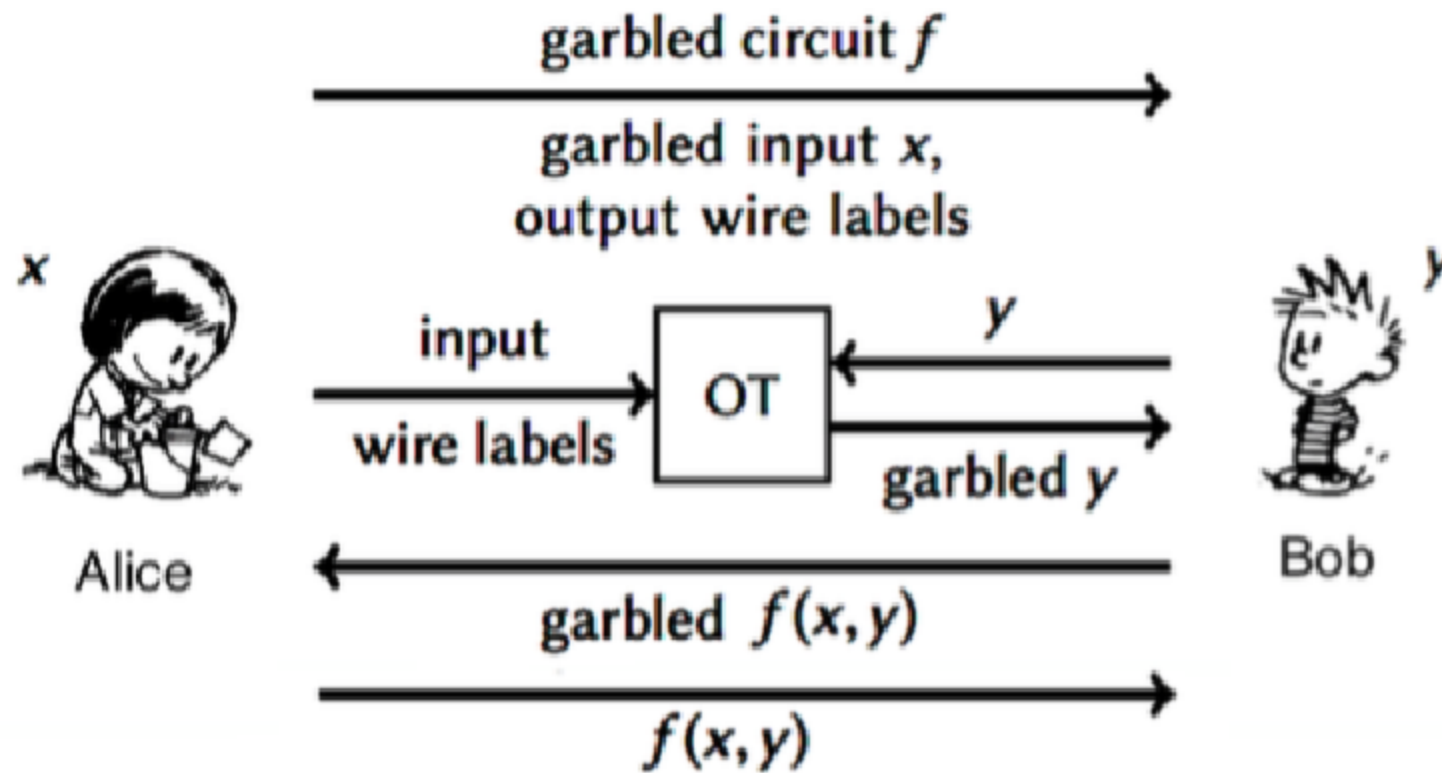
GARBLED CIRCUITS



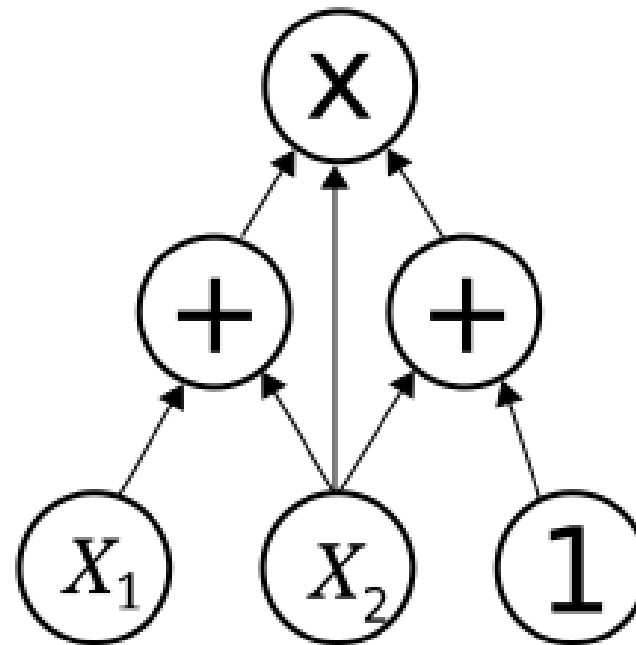
GARBLED CIRCUITS



GARBLED CIRCUITS



ARITHMETIC CIRCUITS



ADDITIVE SECRET SHARING



Gen(): Agree on a group \mathbb{G} with group operation $+$ and its inverse operation $-$ and set $\text{pp} \leftarrow \mathbb{G}$. Return pp .

Share(pp, x, n, t): Parse pp as \mathbb{G} . If $n \neq t$ or $x \notin \mathbb{G}$, return \perp . Otherwise, for $i \in \{1, \dots, n-1\}$ set $x_i \xleftarrow{\$} \mathbb{G}$. Set $x_n = x - \sum_{i=1}^{n-1} x_i$. Return $\{x_1, \dots, x_n\}$.

Reconstruct(pp, $\{x_1, \dots, x_t\}$): For all $i \in [t]$, if there exists an x_i such that $x_i \notin \mathbb{G}$, return \perp . Otherwise, return $x = \sum_{i=1}^t x_i$.

Scheme 2: Additive Secret Sharing.

ADDITIVE SECRET SHARING



for a given variable x , we denote by $\langle x \rangle$ the **shared** form of x

$\langle z \rangle = \langle x \rangle + \langle y \rangle$ means **adding the share** x_i to the share y_i and calling the result z_i for all $i \in [n]$.). The same holds for multiplications with public

values (e.g., $\langle z \rangle = 3 \cdot \langle x \rangle$ means **multiplying the share** x_i by 3 and calling the result z_i for all $i \in [n]$.), whereas additions with public values are only

added to the first share and not to all shares (e.g., $\langle z \rangle = \langle x \rangle + 3$ means adding 3 to the **share x_1 only**).

BEAVER TRIPPLES



Algorithm 3 Multiplying two secret values (Multiplication)

Data: Shared secrets $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$

Result: Shared result $\llbracket w \rrbracket$, where $w = x \cdot y$

- 1: \mathcal{CP} collaboratively choose a triple $\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket$, where $c = a \cdot b$
 - 2: \mathcal{CP} compute $\llbracket e \rrbracket = \llbracket x \rrbracket - \llbracket a \rrbracket$ and $\llbracket d \rrbracket = \llbracket y \rrbracket - \llbracket b \rrbracket$
 - 3: \mathcal{CP} collaboratively open $\llbracket e \rrbracket$ and $\llbracket d \rrbracket$ to all \mathcal{CP}
 - 4: \mathcal{CP} compute $\llbracket w \rrbracket = \llbracket c \rrbracket + e \cdot \llbracket b \rrbracket + d \cdot \llbracket a \rrbracket + e \cdot d$
 - 5: **return** $\llbracket w \rrbracket$
-

Addition mit einer
Konstanten ist ja
nur beim ersten
Term!

BEAVER TRIPPLES



Algorithm 5 Verifying the correctness of multiplicative triples

Data: Secret shared random triple $\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket w \rrbracket$

Result: *True* if $w = x \cdot y$, *False* in case any check fails

- 1: \mathcal{CP} collaboratively choose a random value $\llbracket r \rrbracket$ and open it to all parties
 - 2: \mathcal{CP} collaboratively choose a triple $\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket$, where presumably $c = a \cdot b$
 - 3: \mathcal{CP} compute $\llbracket e \rrbracket = r \cdot \llbracket x \rrbracket - \llbracket a \rrbracket$ and $\llbracket d \rrbracket = \llbracket y \rrbracket - \llbracket b \rrbracket$
 - 4: \mathcal{CP} open $\llbracket e \rrbracket$ and $\llbracket d \rrbracket$ to all \mathcal{CP}
 - 5: \mathcal{CP} compute $\llbracket h \rrbracket = r \cdot \llbracket w \rrbracket - \llbracket c \rrbracket - e \cdot \llbracket b \rrbracket - d \cdot \llbracket a \rrbracket - e \cdot d$
 - 6: \mathcal{CP} open $\llbracket h \rrbracket$ to all \mathcal{CP}
 - 7: **return** $h == 0$
-

Falls $w = x \cdot y + r_1$ und $c = a \cdot b + r_2$, dann ist $h = r \cdot r_1 - r_2$

Initialize: The parties first invoke the preprocessing to get the shared secret key $[\alpha]$, a sufficient number of multiplication triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, and pairs of random values $\langle r \rangle, [r]$, as well as single random values $[t], [e]$.

Then the steps below are performed in sequence according to the structure of the circuit to compute.

Input: To share P_i 's input x_i , P_i takes an available pair $\langle r \rangle, [r]$. Then, do the following:

1. $[r]$ is opened to P_i (if it is known in advance that P_i will provide input, this step can be done already in the preprocessing stage).
2. P_i broadcasts $\epsilon \leftarrow x_i - r$.
3. The parties compute $\langle x_i \rangle \leftarrow \langle r \rangle + \epsilon$.

Add: To add two representations $\langle x \rangle, \langle y \rangle$, the parties locally compute $\langle x \rangle + \langle y \rangle$.

Multiply: To multiply $\langle x \rangle, \langle y \rangle$ the parties do the following:

1. They take two triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle), (\langle f \rangle, \langle g \rangle, \langle h \rangle)$ from the set of the available ones and check that indeed $a \cdot b = c$.
 - Open a representation of a random value $[t]$.
 - partially open $t \cdot \langle a \rangle - \langle f \rangle$ to get ρ and $\langle b \rangle - \langle g \rangle$ to get σ
 - evaluate $t \cdot \langle c \rangle - \langle h \rangle - \sigma \cdot \langle f \rangle - \rho \cdot \langle g \rangle - \sigma \cdot \rho$, and partially open the result.
 - If the result is not zero the players abort, otherwise go on with $\langle a \rangle, \langle b \rangle, \langle c \rangle$.

Note that this check could in fact be done as part of the preprocessing. Moreover, it can be done for all triples in parallel, and so we actually need only one random value t .

2. The parties partially open $\langle x \rangle - \langle a \rangle$ to get ϵ and $\langle y \rangle - \langle b \rangle$ to get δ and compute $\langle z \rangle \leftarrow \langle c \rangle + \epsilon \langle b \rangle + \delta \langle a \rangle + \epsilon \delta$

Output: We enter this stage when the players have $\langle y \rangle$ for the output value y , but this value has been not been opened (the output value is only correct if players have behaved honestly). We then do the following:

1. Let a_1, \dots, a_T be all values publicly opened so far, where $\langle a_j \rangle = (\delta_j, (a_{j,1}, \dots, a_{j,n}), (\gamma(a_j)_1, \dots, \gamma(a_j)_n))$. Now, a random value $[e]$ is opened, and players set $e_i = e^i$ for $i = 1, \dots, T$. All players compute $a \leftarrow \sum_j e_j a_j$.
2. Each P_i calls \mathcal{F}_{COM} to commit to $\gamma_i \leftarrow \sum_j e_j \gamma(a_j)_i$. For the output value $\langle y \rangle$, P_i also commits to his share y_i , and his share $\gamma(y)_i$ in the corresponding MAC.
3. $[\alpha]$ is opened.
4. Each P_i asks \mathcal{F}_{COM} to open γ_i , and all players check that $\alpha(a + \sum_j e_j \delta_j) = \sum_i \gamma_i$. If this is not OK, the protocol aborts. Otherwise the players conclude that the output value is correctly computed.
5. To get the output value y , the commitments to $y_i, \gamma(y)_i$ are opened. Now, y is defined as $y := \sum_i y_i$ and each player checks that $\alpha(y + \delta) = \sum_i \gamma(y)_i$, if so, y is the output.

Fig. 1. The online phase.



SPDZ

(Ivan Damgård, Valerio Pastro,
Nigel P. Smart, and Sarah
Zakarias, 2012)

ZERO KNOWLEDGE



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



ZERO KNOWLEDGE



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

„Kann man beweisen, dass man älter als 18 Jahre ist, ohne sein Geburtsdatum preiszugeben?“

„Kann man beweisen, dass man kreditwürdig ist, ohne seinen Kontostand preiszugeben?“

„Kann man beweisen, dass man die Lösung für ein Problem hat, ohne die Lösung preiszugeben?“





ZERO KNOWLEDGE EIGENSCHAFTEN

- Complete
- Sound
- Zero Knowledge

Also wirklich NICHTS anderes!

*Der „verifier“ lernt nichts anderes, außer der Tatsache,
dass die Aussage wahr ist*

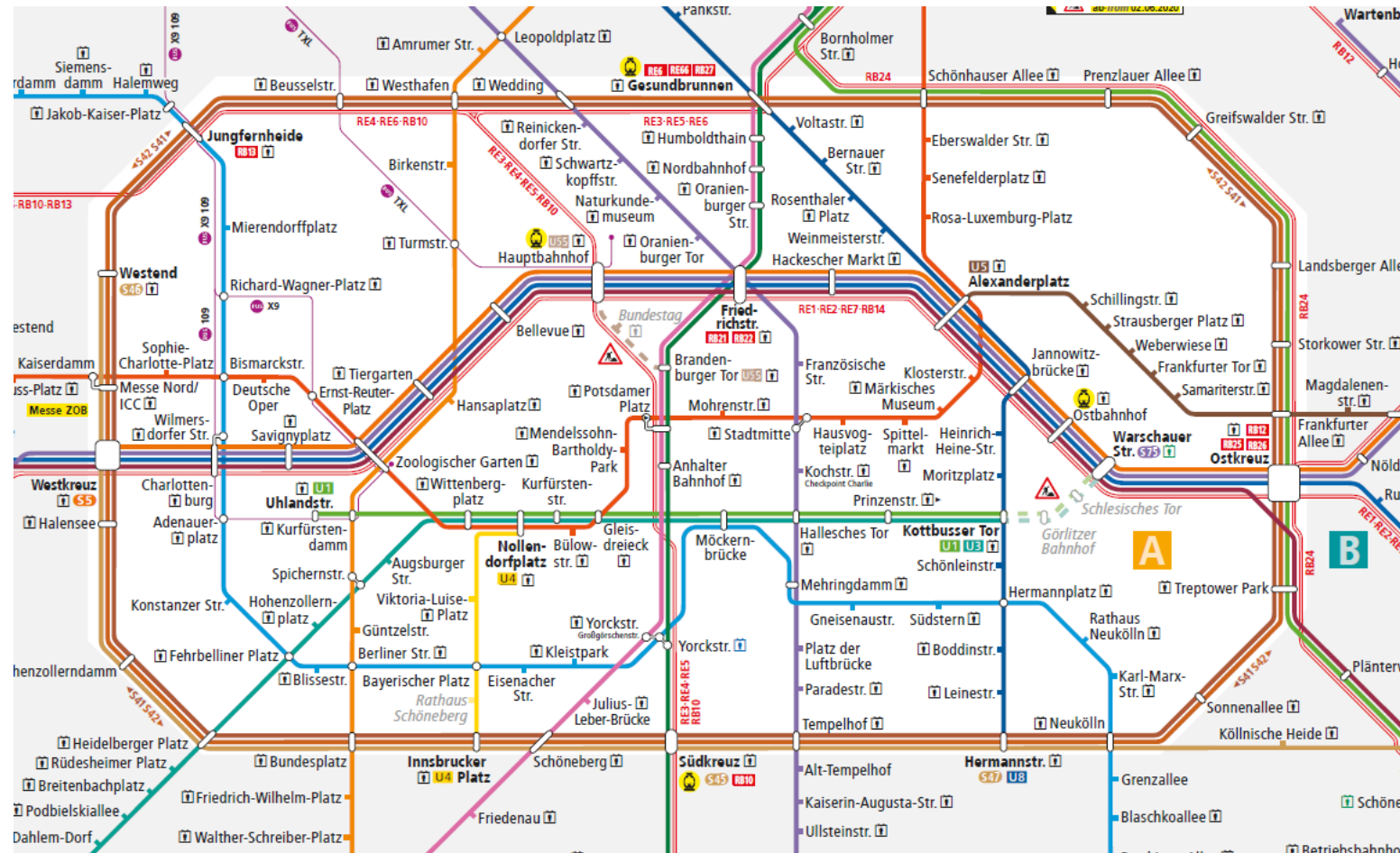
ZERO KNOWLEDGE



Zero Knowledge Proof

1. **Completeness:** if the statement is true, an honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
2. **Soundness:** if the statement is false, no cheating prover can convince an honest verifier that it is true, except with some small probability.
3. **Zero-knowledge:** if the statement is true, no verifier learns anything other than the fact that the statement is true. In other words, just knowing the statement (not the secret) is sufficient to imagine a scenario showing that the prover knows the secret. This is formalized by showing that every verifier has some **simulator** that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between an honest prover and the verifier in question.

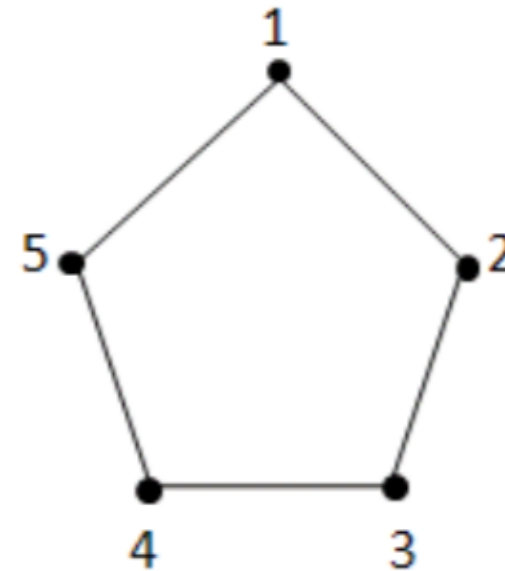
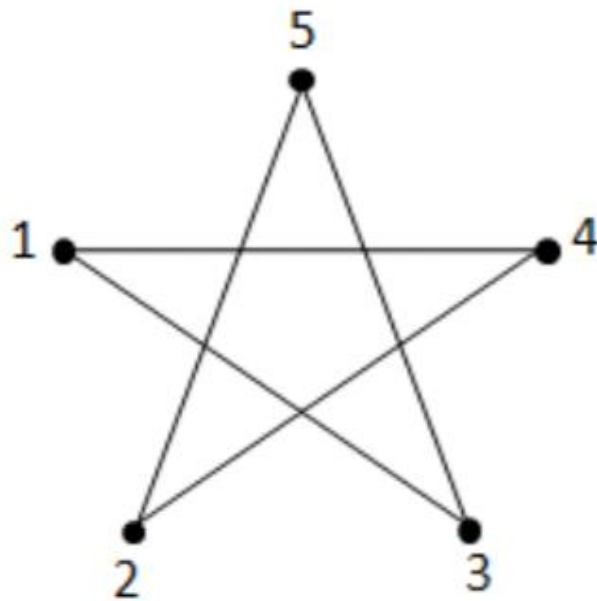
ZERO KNOWLEDGE



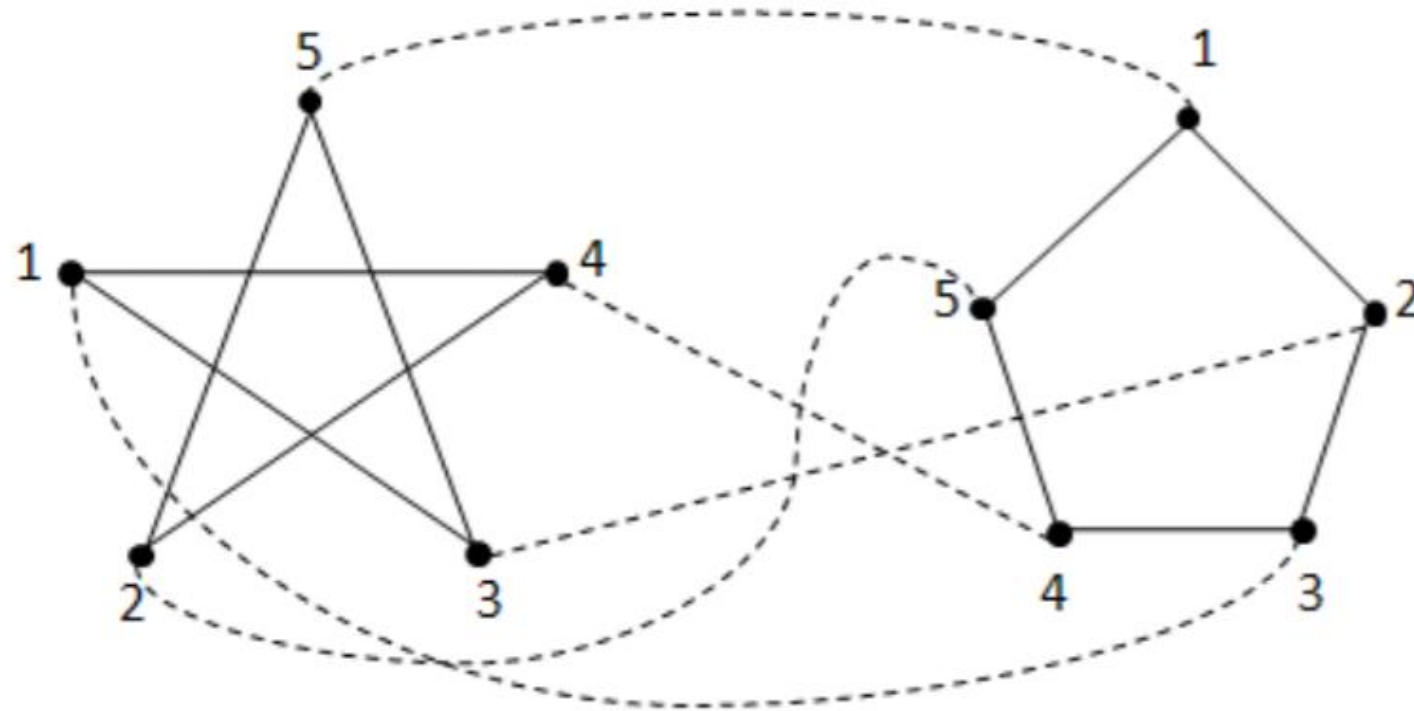
ZERO KNOWLEDGE



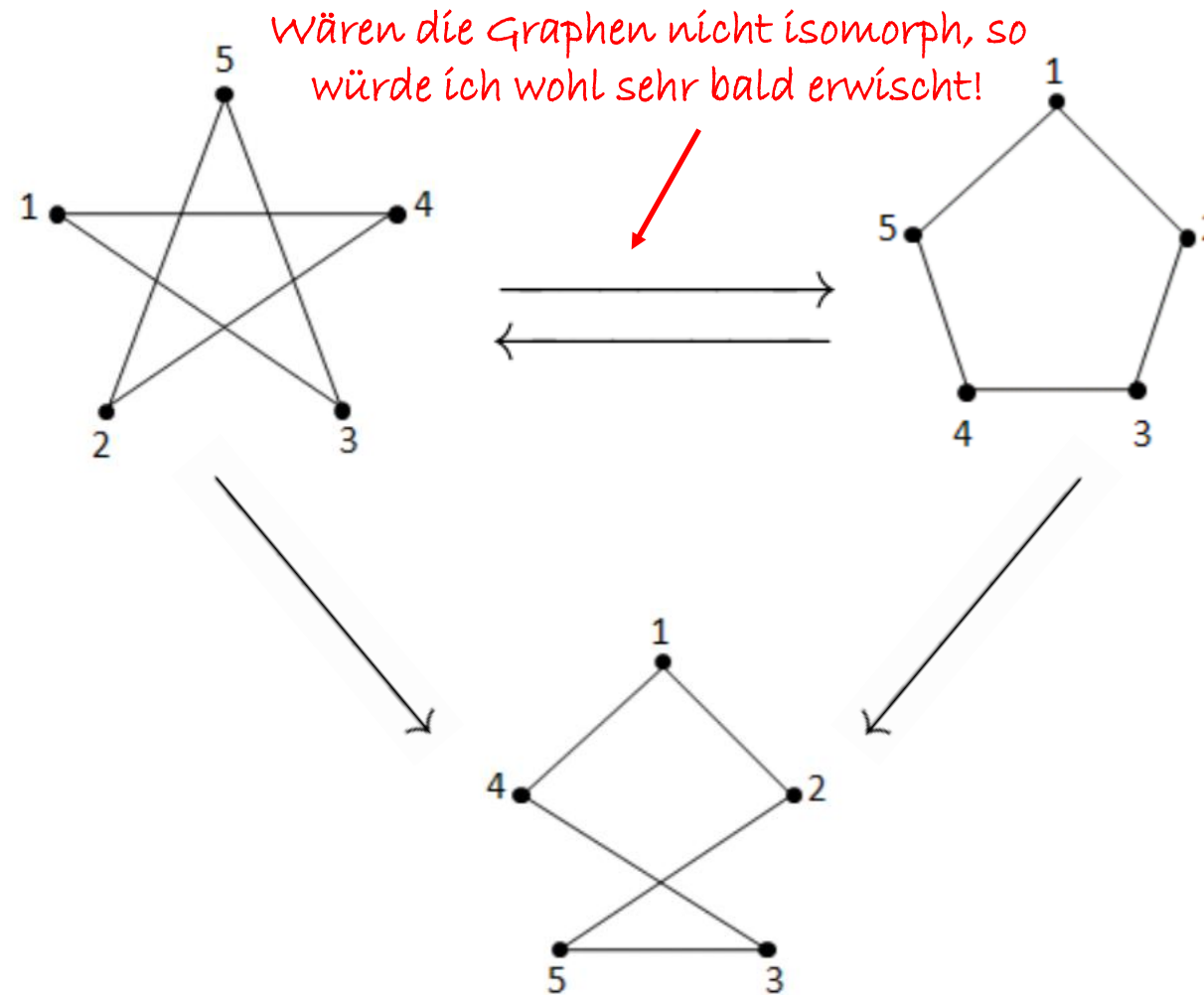
*Sind die beiden Graphen
isomorph?*



ZERO KNOWLEDGE



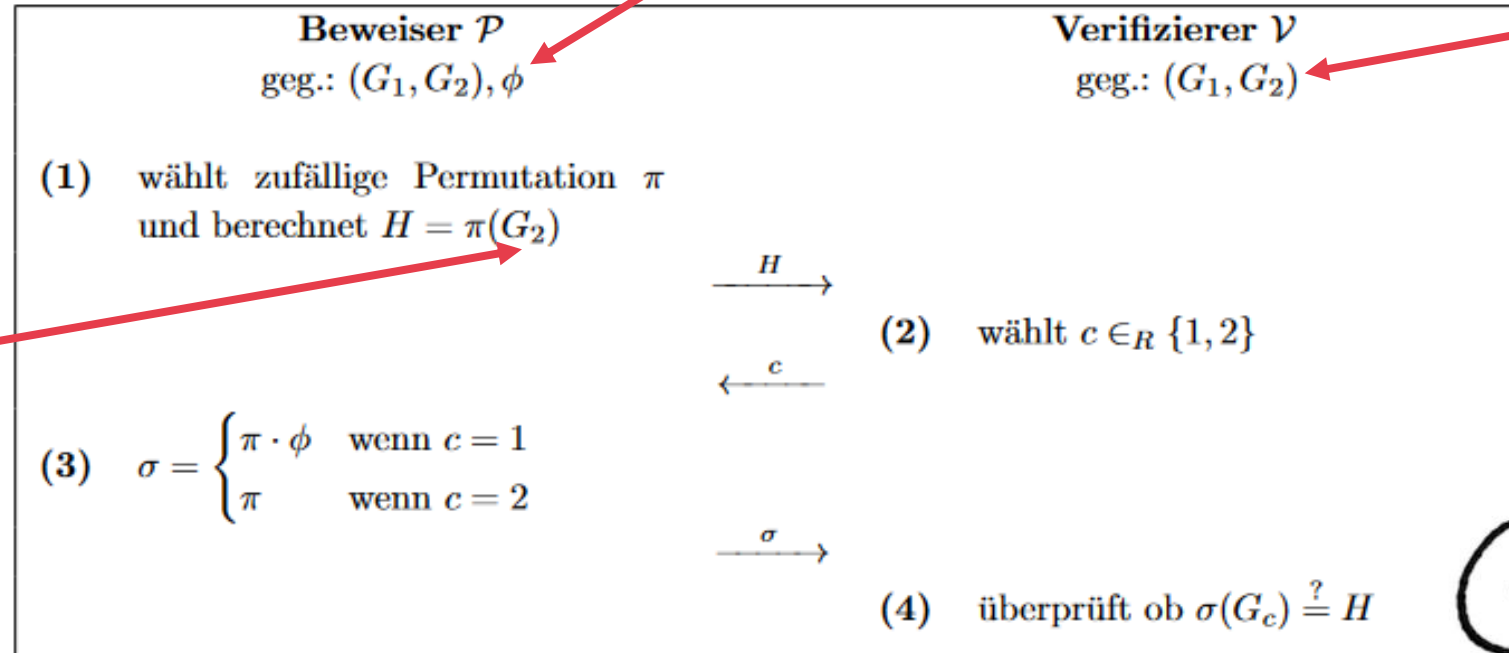
ZERO KNOWLEDGE



ZERO KNOWLEDGE



Graph-Isomorphism



\mathcal{P} kann sich in
jeder Runde neu
aussuchen,
welchen Graph er
wählt...

Secret

Public



ZERO KNOWLEDGE



Quadratic

Residues

Secret

Public

Schlüsselgenerierung	
Sei $n = p \cdot q$ das Produkt zweier zufälliger, verschiedener Primzahlen	
Der Beweiser \mathcal{P} wählt $w \in_R \mathbb{Z}_n^*$ und berechnet $x \equiv w^2 \pmod n$. Die Werte (n, x) werden als öffentlicher Schlüssel veröffentlicht, w bildet das Geheimnis von \mathcal{P} .	
Protokoll	
Beweiser geg.: $(n, x)w$	Verifizierer geg.: (n, x)
wählt s zufällig in \mathbb{Z}_n^* berechnet $a = s^2 \pmod n$	
	\xrightarrow{a}
	wählt $c \in_R \{0, 1\}$
	\xleftarrow{c}
berechnet $z = s \cdot w^c \pmod n$	
	\xrightarrow{z}
	überprüft ob $z^2 \stackrel{?}{=} a \cdot x^c \pmod n$



ZERO KNOWLEDGE



Diskrete

Logarithm

Public

Schlüsselgenerierung	
<p>gegebene Parameter: Primzahlen p, q mit $q (p-1)$, Generator g der Untergruppe G_q</p> <p>Beweiser wählt w zufällig in \mathbb{Z}_q und erzeugt sein Schlüsselpaar – bestehend aus einem öffentlichen Schlüssel x und einem geheimen Schlüssel w – durch:</p>	
$(x = g^w \bmod p, \quad w)$	
Protokoll	
Beweiser \mathcal{P} geg.: $(p, q, g) \quad x, w$	Verifizierer \mathcal{V} geg.: $(p, q, g) \quad x$
wählt s zufällig in \mathbb{Z}_q berechnet $a = g^s \bmod p$	
	wählt $c \in_R \{0, 1\}$
berechnet $z = s + c \cdot w \bmod q$	
	überprüft ob $g^z \stackrel{?}{=} a \cdot x^c \bmod p$

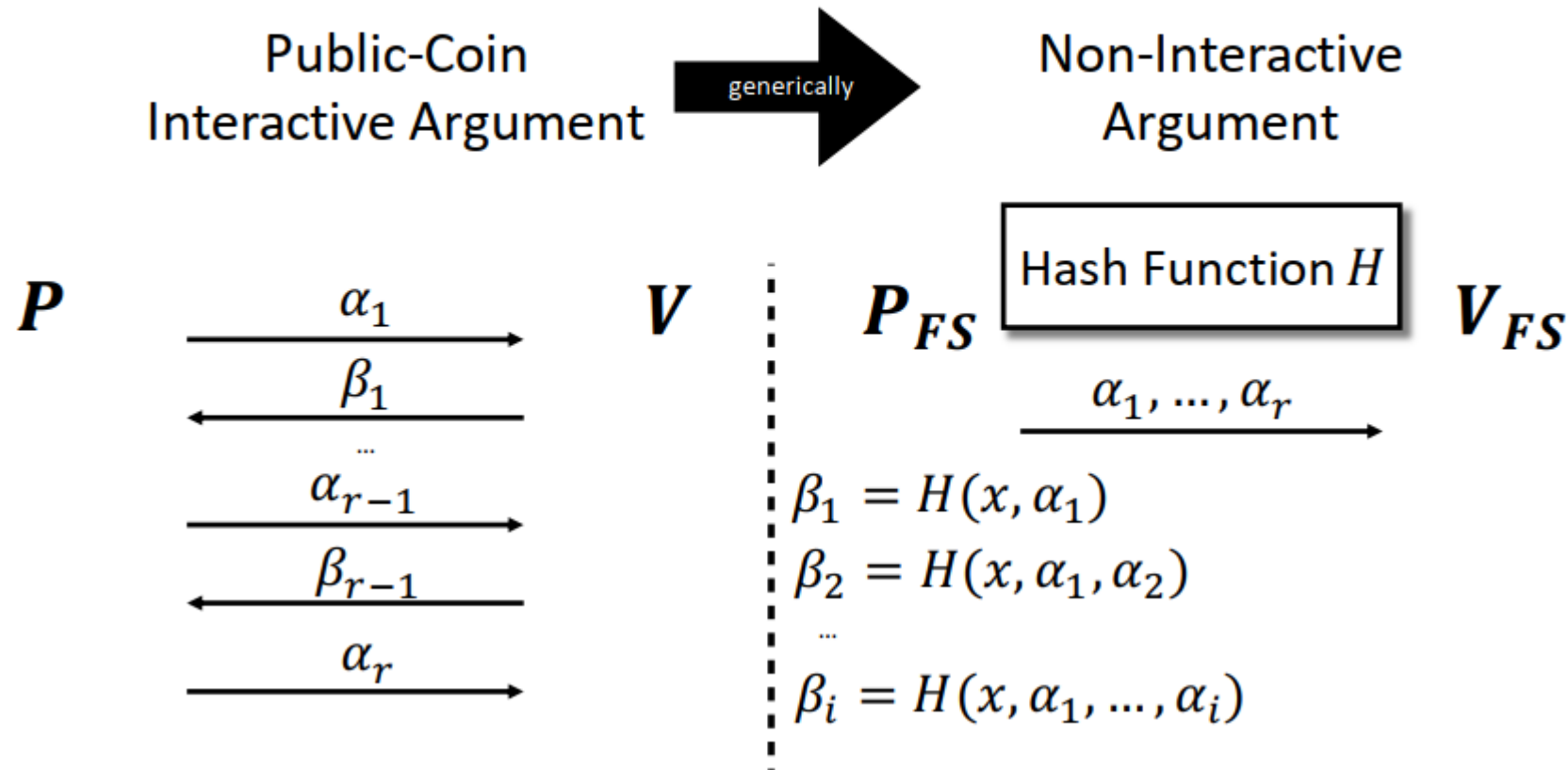
Secret



ZERO KNOWLEDGE



The Fiat-Shamir Transform



ZERO KNOWLEDGE



Signature Scheme

Schlüsselgenerierung:

gegebene Parameter: Primzahlen p, q mit $q|(p-1)$, Generator g der Untergruppe G_q , Einweg-Hashfunktion H

Beweiser wählt $w \in \mathbb{Z}_q$ und veröffentlicht den Schlüssel $x = g^w \bmod p$

Public

Secret

Signieren einer Nachricht m :

- (1) Beweiser wählt s_1, \dots, s_k zufällig in \mathbb{Z}_q und berechnet $a_i = g^{s_i} \bmod p$ für $i = 1, \dots, k$
- (2) Beweiser berechnet $H(m||a_1||\dots||a_k)$ und nutzt die ersten k Bits des Hashwertes als Challenges c_1, \dots, c_k
- (3) Beweiser berechnet Antworten $z_i = s_i + c_i \cdot w \bmod q$ für $i = 1, \dots, k$
- (4) Beweiser sendet Nachricht m und zugehörige Signatur $(c_1, \dots, c_k, z_1, \dots, z_k)$

Verifizieren der Signatur $(c_1, \dots, c_k, z_1, \dots, z_k)$ für Nachricht m :

- (1) Verifizierer berechnet $b_i = g^{z_i} \cdot x^{-c_i} \bmod p$ für $i = 1, \dots, k$
- (2) Verifizierer ermittelt $H(m||b_1||\dots||b_k)$
- (3) Verifizierer akzeptiert, wenn die ersten k Bits von $H(m||b_1||\dots||b_k)$ identisch mit c_1, \dots, c_k der Signatur sind.



ZERO KNOWLEDGE



MPC in the Head

Let L be a language in NP and let $R(x, w)$ be a corresponding NP-relation. Let f be the following $(n + 1)$ -argument function ($n \geq 3$), corresponding to R :

$$f(x, w_1, \dots, w_n) = R(x, w_1 \oplus \dots \oplus w_n),$$

where \oplus here denotes bitwise exclusive-or of strings (all of the same length). We view f as an n -party functionality, where the first argument x is a public input known to all n players, w_i is a private input of player P_i , and the output is received by all players. We will sometimes ignore the public input x , viewing f as an n -argument function specified by x .

ZERO KNOWLEDGE



MPC in the Head

Π_f be an n -party protocol

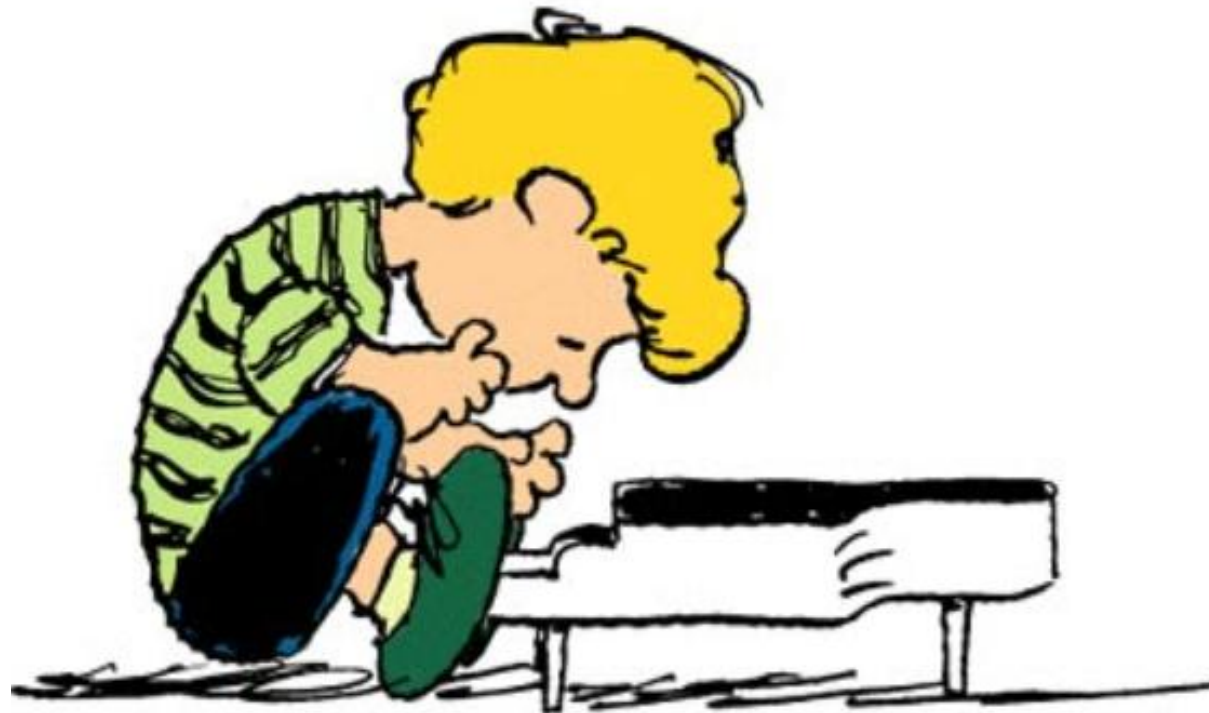
Zero-knowledge protocol Π_R in the commitment-hybrid model

1. The prover picks at random $w_1, \dots, w_n \in \{0, 1\}^m$ whose exclusive-or equals the witness w . She emulates “in her head” the execution of Π_f on input (x, w_1, \dots, w_n) (this involves choosing randomness for the n players and running the protocol). Based on this execution, the prover prepares the views V_1, \dots, V_n of the n players; she separately commits to each of these n views.
2. Verifier picks at random distinct player indices $i, j \in [n]$ and sends them to the prover.
3. Prover “opens” the commitments corresponding to the two views V_i, V_j .
4. Verifier accepts if and only if:
 - (a) the prover indeed successfully opened the two requested views,
 - (b) the outputs of both P_i and P_j (which are determined by their views) are 1, and
 - (c) the two opened views are consistent with each other

DIFFERENTIAL PRIVACY



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



DIFFERENTIAL PRIVACY



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



„Wie bringt man Marcie dazu, eine
Tabu-Frage wahrheitsgemäß zu beantworten?“



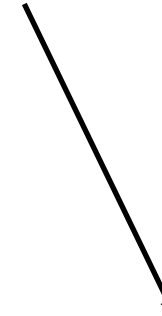
DIFFERENTIAL PRIVACY



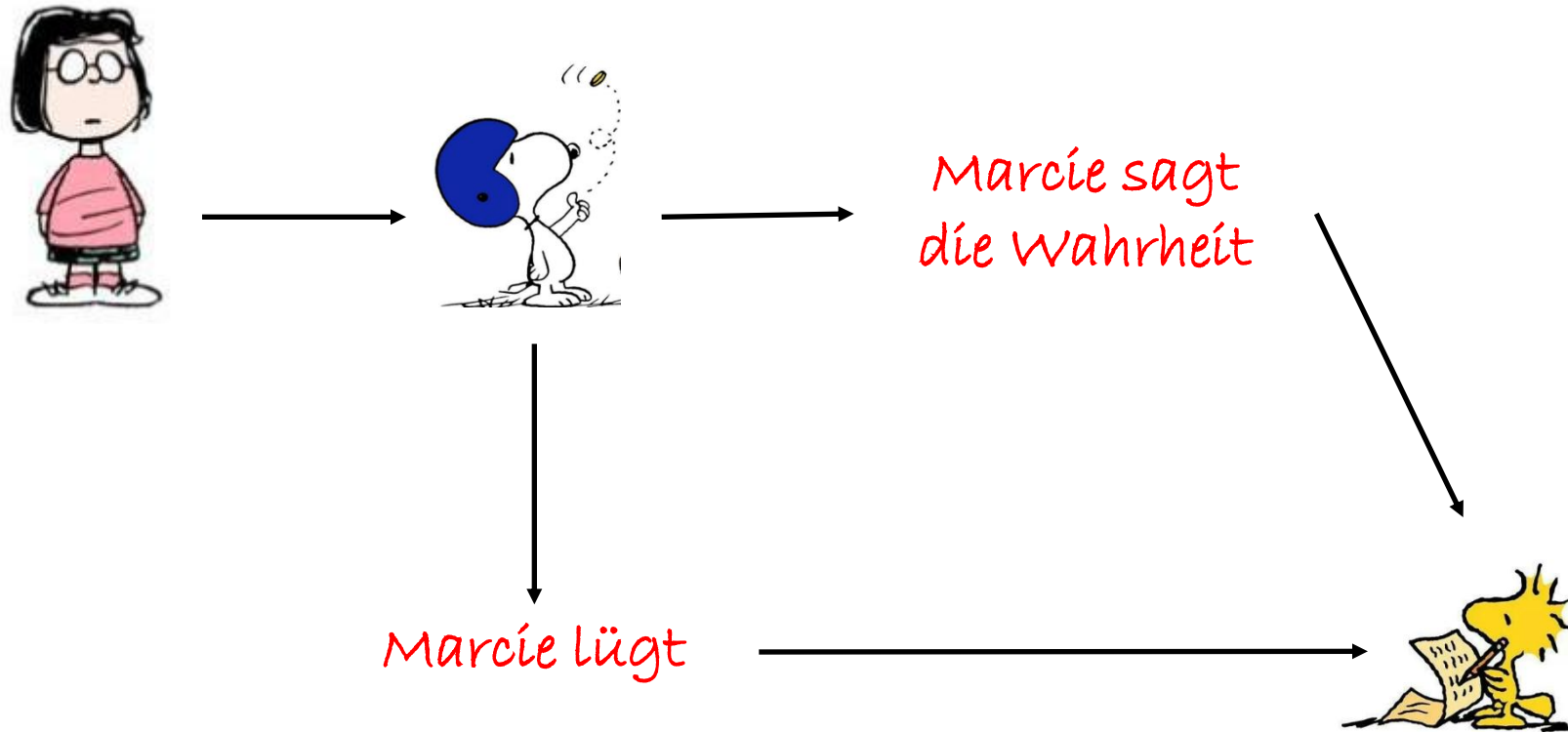
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



*Marcie sagt
die Wahrheit*



DIFFERENTIAL PRIVACY



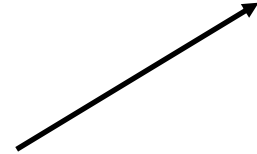
DIFFERENTIAL PRIVACY



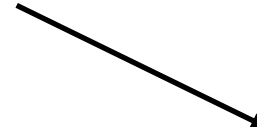
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law



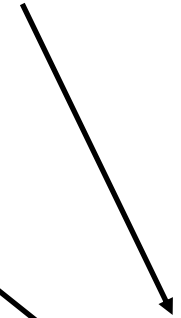
Marcie sagt
die Wahrheit



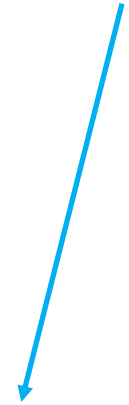
Marcie sagt
„Ja“



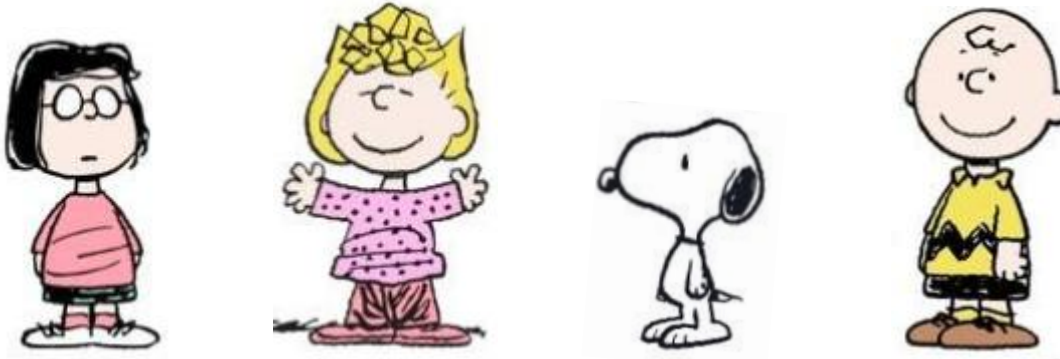
Marcie sagt
„Nein“



In $\frac{3}{4}$ der Fälle steht bei Marcie die richtige Antwort, in $\frac{1}{4}$ die falsche



DIFFERENTIAL PRIVACY



Das Rauschen lässt sich wieder rausrechnen:

Prozent tatsächliche „Ja“-Antworten = $2(\text{Prozent „Ja“-Antworten} - 0,25)$

$$\frac{3}{4} \text{ mal } 25\% = 18,75\%$$

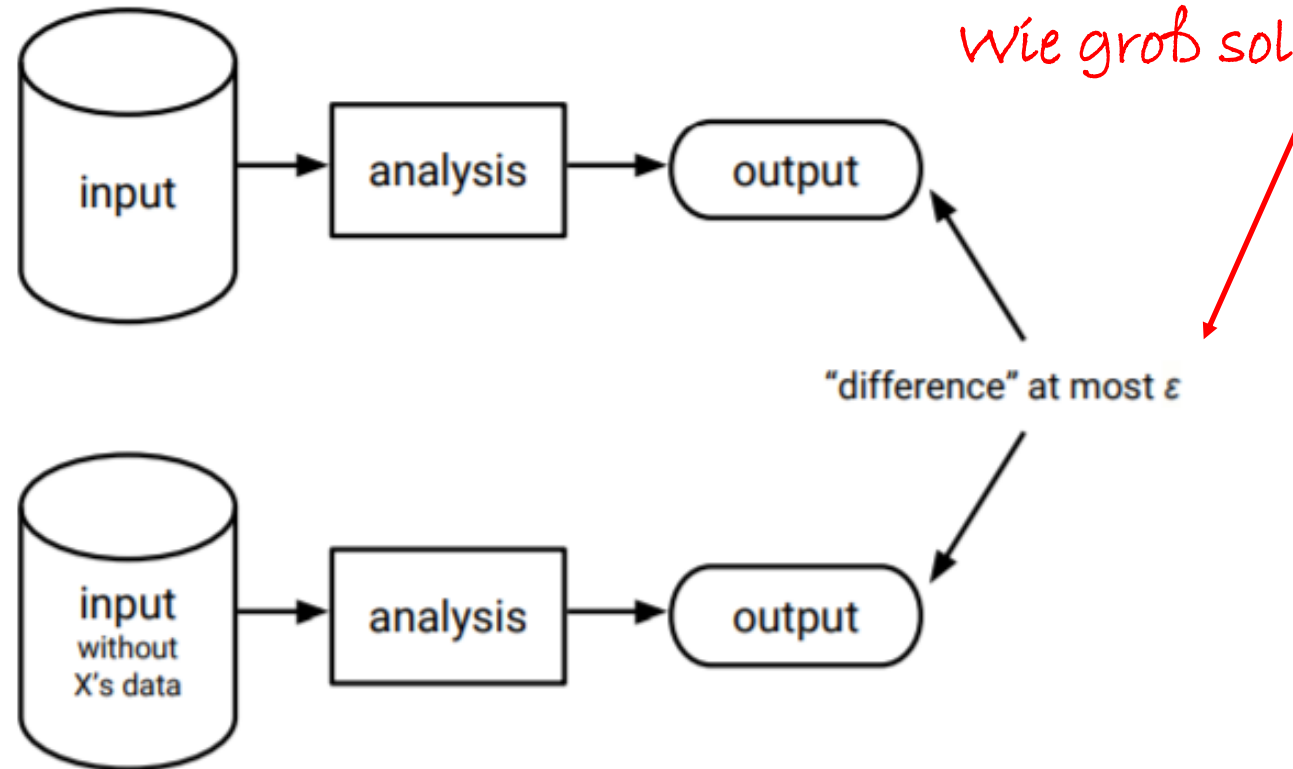
$$\frac{1}{4} \text{ mal } 75\% = 18,75\%$$

$$37,5\%$$

$$2(0,375 - 0,25) = 0,25$$



DIFFERENTIAL PRIVACY



DIFFERENTIAL PRIVACY



A function which satisfies **differential privacy** is often called a *mechanism*. We say that a **mechanism** F satisfies differential privacy if for all **neighboring datasets** x and x' , and all possible outputs S ,

$$\frac{\Pr[F(x) = S]}{\Pr[F(x') = S]} \leq e^\epsilon \quad (4.1)$$

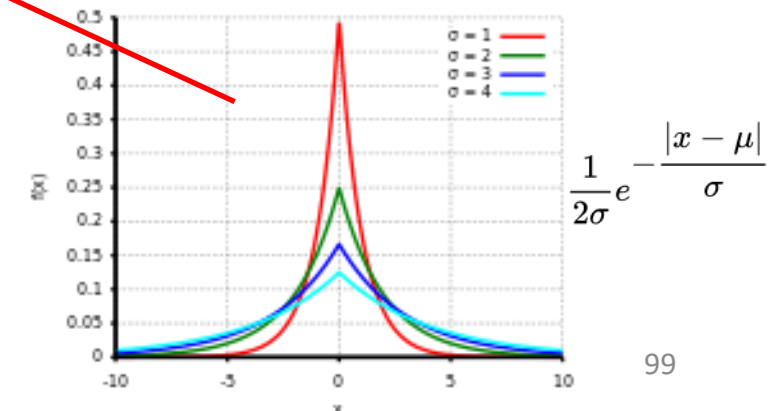
According to the **Laplace** mechanism, for a function $f(x)$ which returns a number, the following definition of $F(x)$ satisfies ϵ -differential privacy:

$$F(x) = f(x) + \text{Lap}\left(\frac{s}{\epsilon}\right) \quad (4.2)$$

where s is the **sensitivity** of f , and $\text{Lap}(S)$ denotes sampling from the **Laplace distribution** with center 0 and scale S .

Gibt die Größe der Änderung des Funktionswertes an
bei Erhöhung der Eingabe um 1

$$GS(f) = \max_{x, x': d(x, x') \leq 1} |f(x) - f(x')|$$



DIFFERENTIAL PRIVACY



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Approximate differential privacy, also called (ϵ, δ) -differential privacy, has the following definition:

$$\Pr[F(x) = S] \leq e^\epsilon \Pr[F(x') = s] + \delta$$

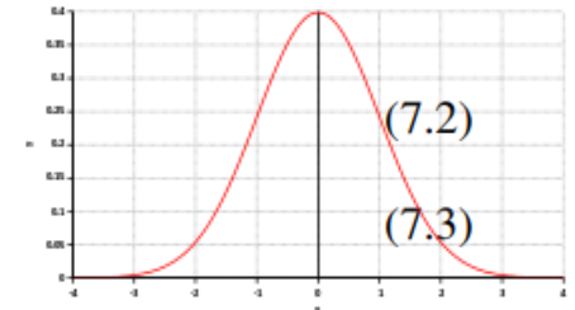
With probability $1 - \delta$, $\frac{\Pr[F(x)=S]}{\Pr[F(x')=s]} \leq e^\epsilon$

With probability δ , we get **no guarantee at all**

According to the **Gaussian mechanism**, for a function $f(x)$ which returns a number, the following definition of $F(x)$ satisfies (ϵ, δ) -differential privacy:

$$F(x) = f(x) + \mathcal{N}(\sigma^2)$$

where $\sigma^2 = \frac{2s^2 \log(1.25/\delta)}{\epsilon^2}$



where s is the sensitivity of f , and $\mathcal{N}(\sigma^2)$ denotes sampling from the **Gaussian (normal) distribution** with center 0 and variance σ^2 . Note that here (and elsewhere in these notes), log denotes the natural logarithm.

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2}$$

DIFFERENTIAL PRIVACY



Drei potentielle
Möglichkeiten
„noise“
hinzuzufügen

Data: Training data set (X, y)

Result: Model parameters θ

$\theta \leftarrow \text{Init}(0)$

#1. Add noise here: *objective perturbation*

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(\theta, X_i, y_i) + \lambda R(\theta) + \beta$$

for *epoch* **in** *epochs* **do**

 #2. Add noise here: *gradient perturbation*

$$\theta = \theta - \eta(\nabla J(\theta) + \beta)$$

end

#3. Add noise here: *output perturbation*

return $\theta + \beta$

Algorithm 1: Privacy noise mechanisms.

TRUSTED EXECUTION ENVIRONMENTS



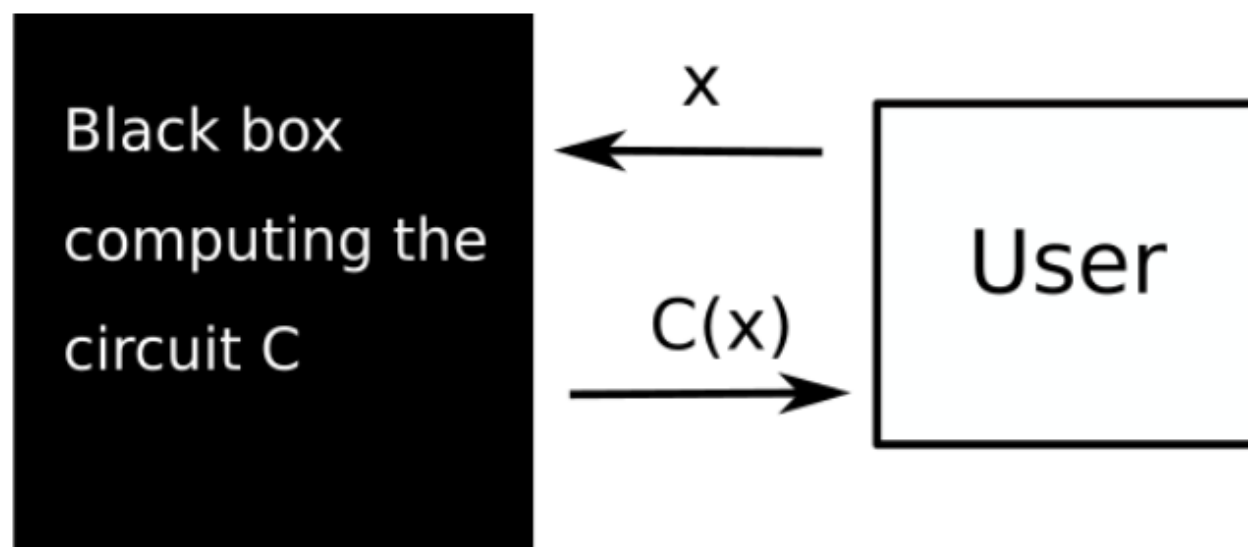
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law





VIRTUAL BLACK BOX OBFUSCATOR

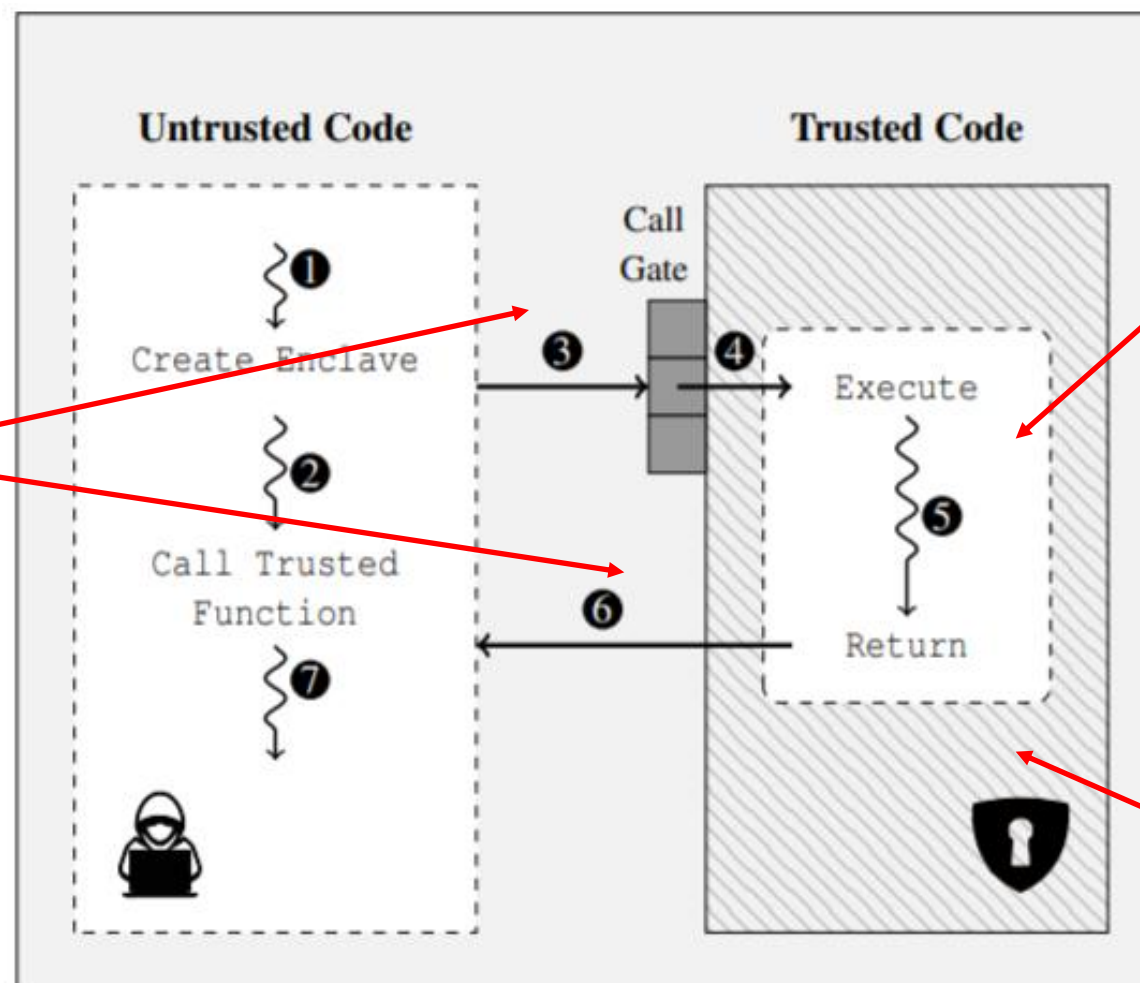
Wir wissen schon, dass der leider nicht existiert...





TRUSTED EXECUTION ENVIRONMENTS

Daten I/O findet ausschließlich
verschlüsselt statt



In der
„enclave“
läuft
attestierter
Code

Selbst ein
Kompro-
mittiertes
OS hat
keinen
Zugriff
auf die
Daten der
„enclave“



PRIVACY ENHANCING TECHNOLOGIES

