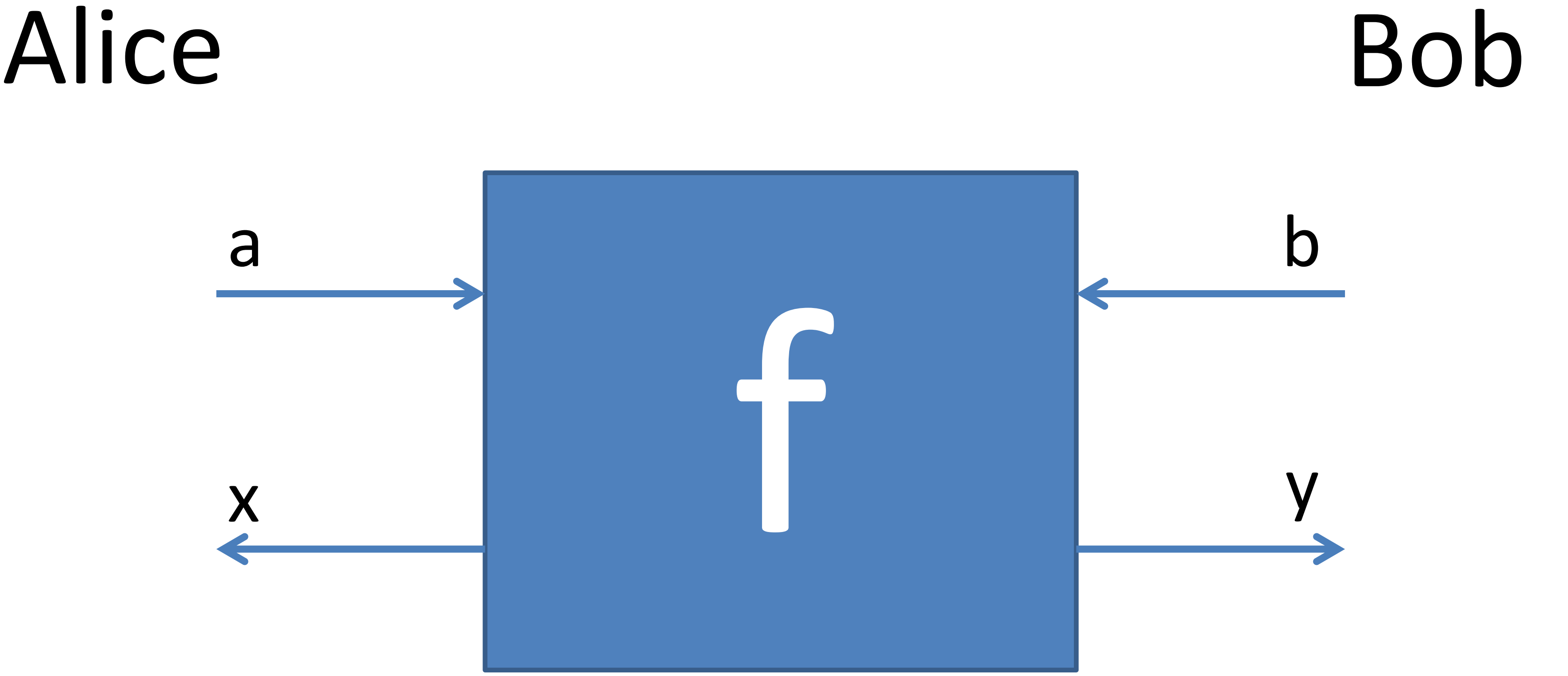
A New Approach to Practical Secure Two-Party Computation

Jesper Buus Nielsen
Peter Sebastian Nordholt
Claudio Orlandi
Sai Sheshank

Secure Two-Party Computation

- Alice has an input $a \in \{0,1\}^*$
- Bob has an input $b \in \{0,1\}^*$
- They agree on a (randomized) function
$$f: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$$
- They want to securely compute
$$(x,y) = f(a,b)$$
- Alice is to learn x and is not allowed to learn any information extra to (a,x)
- Bob is to learn y and is not allowed to learn any information extra to (b,y)

S2C Pictorially



Some Security Flavors

- **Passive**: The protocol is only secure if both parties follow the protocol
- **Active**: The protocol is secure even if one of the parties deviate from the protocol

- **Computational**: Security against poly-time adversaries
- **Unconditional**: The security does not depend on the computational power of the parties

Oblivious Transfer

- S2C of $OT((x_0,x_1), y) = (\varepsilon, x_y)$
  where $x_0, x_1 \in \{0,1\}^k$ and $y \in \{0,1\}$

OT Extension

- OT is provably a public-key primitive
  - OTs can be generated at a rate of 10 to 100 per second depending on the underlying assumption
- OT extension takes a few seed OT and a PRG or hash function and implements any polynomial number of OTs using only a few applications of the symmetric primitive per generated OT
- Like enveloping RSA+AES

OT is Complete

- OTs is complete for cryptography, but most problems in practice are solved using specialized protocols
- Reasons:
  - OT is considered expensive
  - Though there exist practical passive-secure generic protocols based on OT, all active-secure solutions suffer a blowup of k in complexity, where k is the security parameter
  - Thought there exist active-secure protocol asymptotically as efficient as the passive-secure ones, they have enormous constants
- We change this picture

The Result

- We advance the theory of OT-extension, significantly improving the constants
- We implement the improved theory and show that we can generate active-secure OTs at a rate of 500,000 per second
- We improve the theory of basing active-secure two-party computation (S2C) on OTs
  - Asymptotically worse than best previous result
  - Asymptotically better than any result previously implemented
- We implement the theory and show that we can do active-secure S2C at a rate of about 20,000 gates per second
  - Online phase handles 1,000,000 gates per second
  - Online: The part that can only be executed once inputs are known

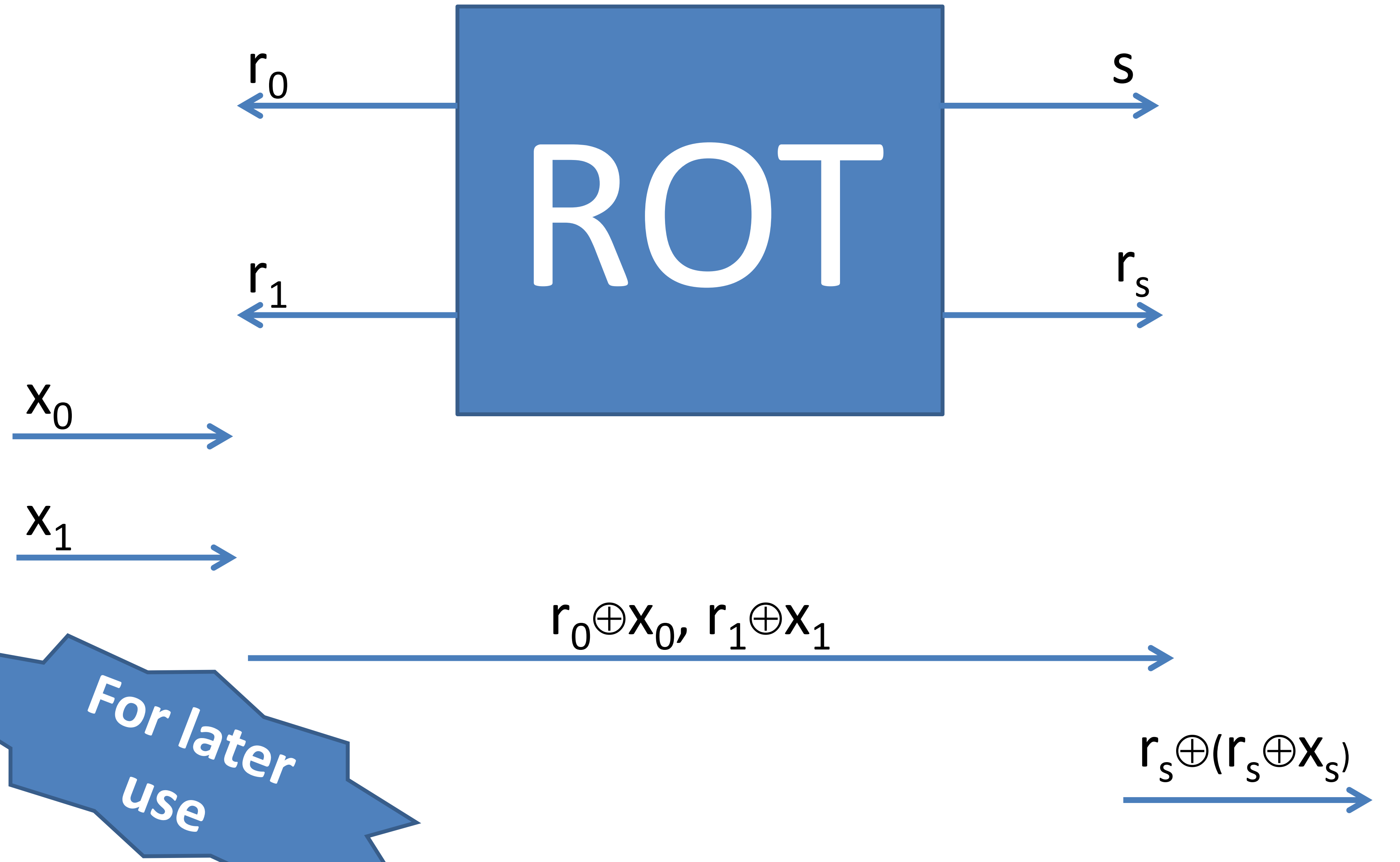
Our Security

- Our protocols are computationally, active secure in the random oracle model
  - We use a PRG
  - Also need a few seed OTs (160)

Random Oblivious Transfer

- S2C of ROT(ε, ε) = $((r_0,r_1), (s, r_s))$
  where $r_0, r_1 \in_R \{0,1\}^k$ and $r \in_R \{0,1\}$

Random Self-Reducibility ROT→OT



Passive-Secure S2P from OT

- A gate-by-gate evaluation of a Boolean circuit computing the function (using Xor + AND)
  - Computing on secret bits
  - Only the outputs are revealed
- **Representation** of a secret bit x:

  A holds $x_A \in \{0,1\}$ B holds $x_B \in \{0,1\}$

  $$x = x_A \oplus x_B$$
- **Input** of some x from A:

  A sets $x_A = x$ B sets $x_B = 0$
- **Output** of some x to A:

  B sends x_B to A

Passive-Secure S2P from OT

- **Representation** of a secret bit x:

  A holds $x_A \in \{0,1\}$ B holds $x_B \in \{0,1\}$

  $$x = x_A \oplus x_B$$

- $x \oplus y = (x_A \oplus x_B) \oplus (y_A \oplus y_B) = (x_A \oplus y_A) \oplus (x_B \oplus y_B)$

- **Xor** secure computation of $z = x \oplus y$:

  A sets $z_A = x_A \oplus y_A$ B sets $z_B = x_B \oplus y_B$

Passive-Secure S2P from OT

- **Representation** of a secret bit x:

  A holds $x_A \in \{0,1\}$ B holds $x_B \in \{0,1\}$

  $$x = x_A \oplus x_B$$

- $xy = (x_A \oplus x_B)(y_A \oplus y_B) = x_A y_A \oplus x_B y_A \oplus x_A y_B \oplus x_B y_B$
- **AND** secure computation of z=xy:

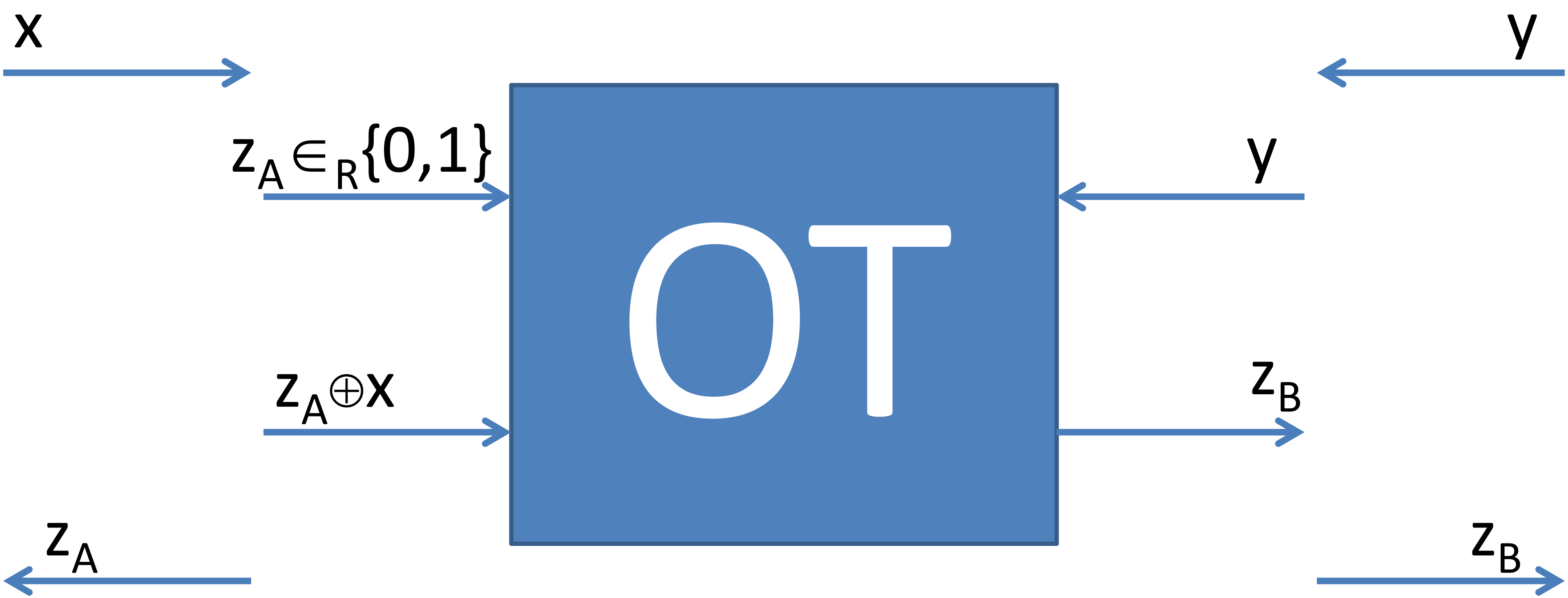
  A sets $t_A = x_A y_A$ B sets $t_B = x_B y_B$

  This is a secure computation of $t = x_A y_A \oplus x_B y_B$

- Then they securely compute $u = x_B y_A$ and $v = x_A y_B$
- Then they securely compute $z = t \oplus u \oplus v$

Secure AND

- S2C of AND(x, y) = (z_A, z_B)
  where $z_A, z_B \in_R \{0,1\}$ and $z_A \oplus z_B = xy$



Passive Security (Only)

- The above protocol is unconditionally passive-secure assuming that all the OTs are unconditionally secure
- The protocol is, however, not active-secure, as a party might deviate at all the points marked with blue with ill effects

Active Security

- To achieve active security, efficiently, we propose to commit both parties to all their shares
- Reminiscent of the notion of committed OT, but we make the crucial difference that we do not base it on (slow) public-key cryptography
- To not confuse with committed OT, we call the technique authenticated OT

Authenticating Alice's Bits

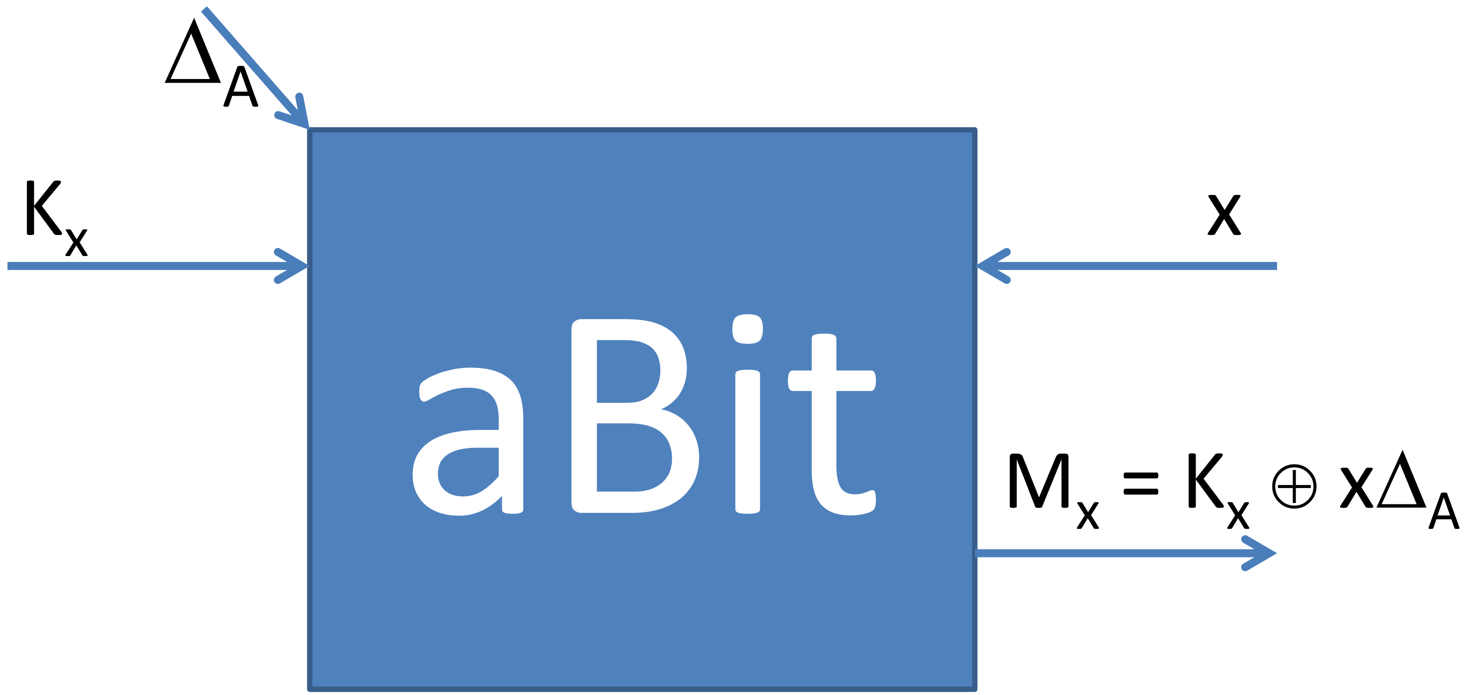
- Alice holds a **global key** $\Delta_A \in_R \{0,1\}^k$
  - k is a security parameter
- For each of Bob's bits x Alice holds a **local key** $K_x \in_R \{0,1\}^k$
- Bob learns only the MAC $M_x = K_x \oplus x\Delta_A$
- Xor-Homomorphic:

Alice:	K_x	Bob:	x	$M_x = K_x \oplus x\Delta_A$
	K_y		y	$M_y = K_y \oplus y\Delta_A$
	$K_z = K_x \oplus K_y$		$z = x \oplus y$	$M_z = M_x \oplus M_y$

Three Little Helpers

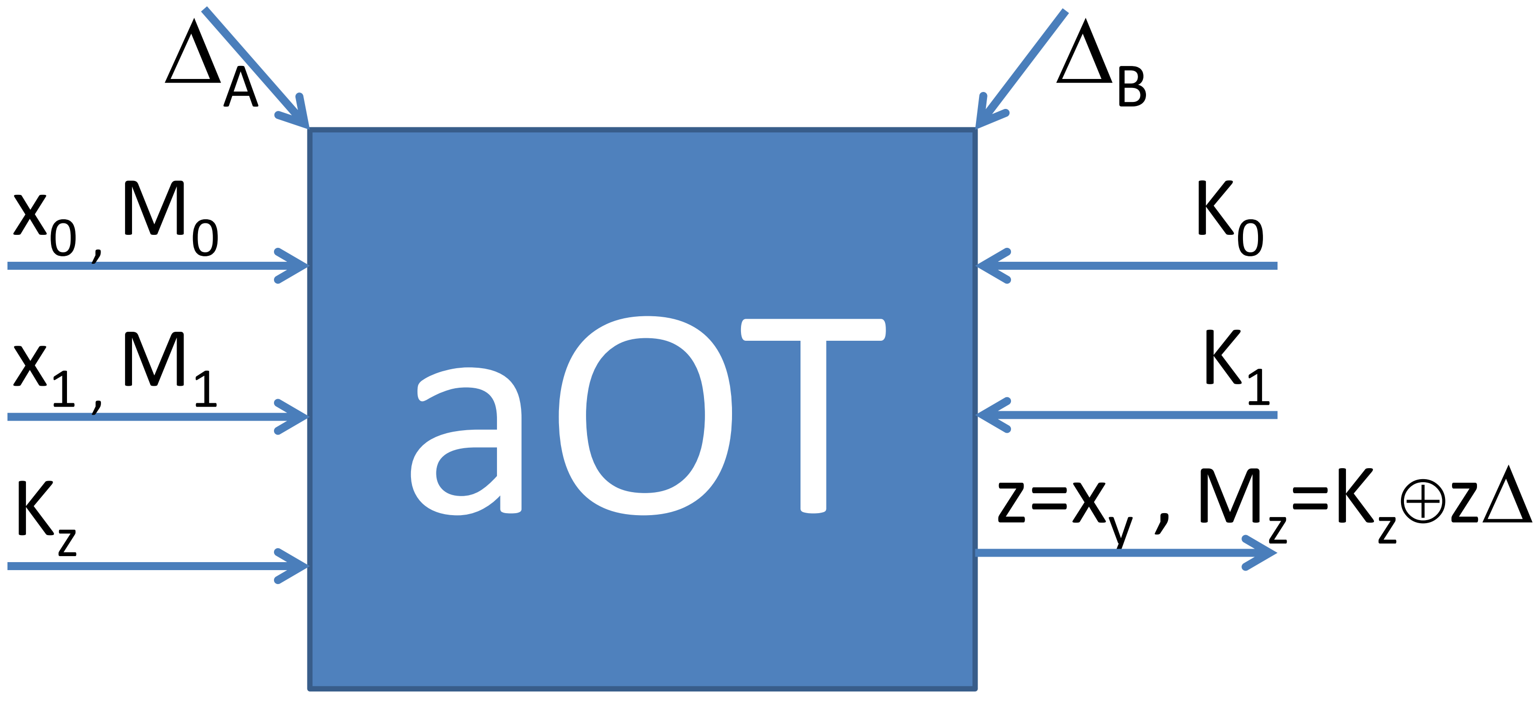
- Next step is to efficiently, actively secure implement three little helpers
- **aBit**: Allows Alice and Bob to authenticate a bit of Bob's using a local key chosen by Alice–the global key is fixed
- **aOT**: Allows Alice and Bob to perform an OT of bits which are authenticated and obtain an authentication on the results
- **aAND**: If Bob holds authenticated x and y, then he can compute z=xy plus an authentication of this result, and only this result
- Similar protocols for the other direction

Authenticating a Bit



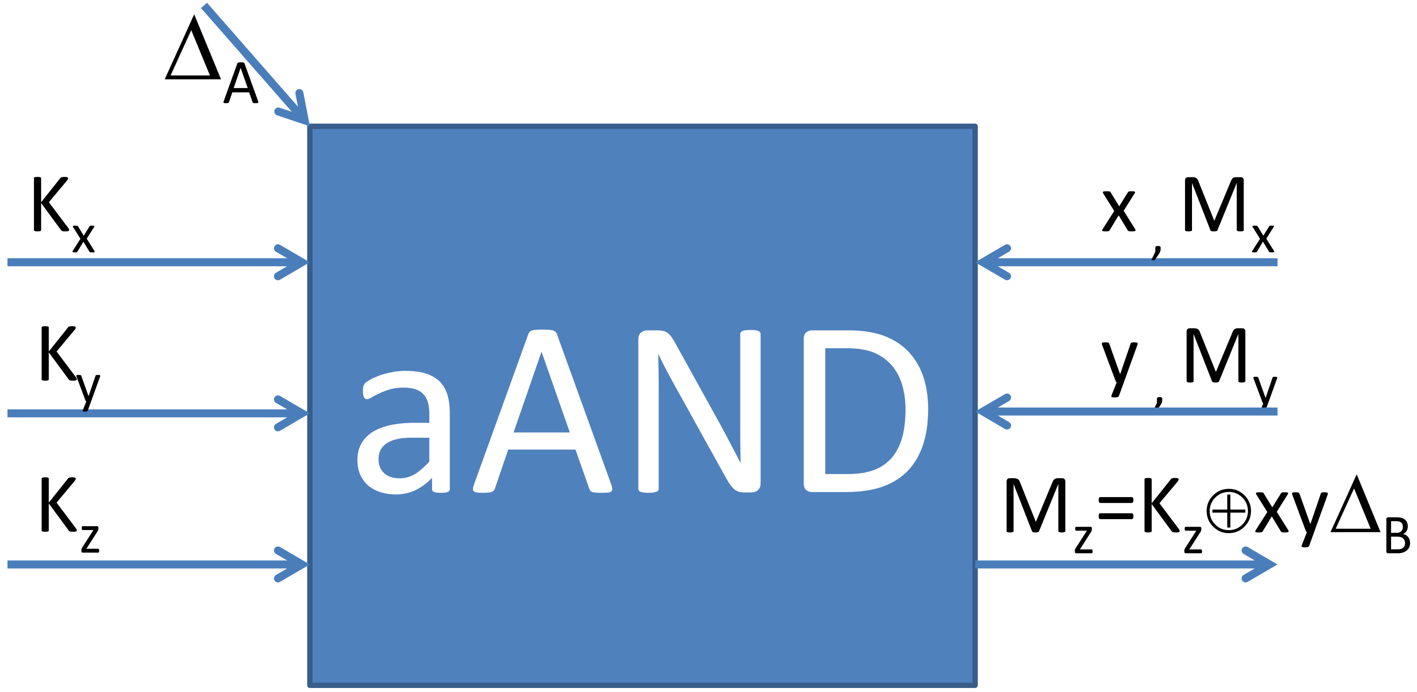
Authenticated Oblivious Transfer

- The protocol outputs *failure* if the MAC are not correct



Authenticated AND

- The protocol outputs *failure* if the MAC are not correct



Active-Secure S2P from OT

- **Representation** of a secret bit x:

  A holds $x_A \in \{0,1\}$ B holds $x_B \in \{0,1\}$

  $$x = x_A \oplus x_B$$

  and both bits are authenticated
- **Input** of some x from A:

  A calls aBit with $x_A = x$ to get it authenticated

  B calls aBit with $x_B = 0$ and sends the MAC as proof
- **Output** of some x to A:

  B sends x_B to A along with the MAC on x_B

Active-Secure S2P from OT

- **Representation** of a secret bit x:
  A holds $x_A \in \{0,1\}$ B holds $x_B \in \{0,1\}$
  $$x = x_A \oplus x_B$$
  and both bits are authenticated

- **Xor** secure computation of $z = x \oplus y$:
  A sets $z_A = x_A \oplus y_A$ B sets $z_B = x_B \oplus y_B$
  They use the Xor-homomorphism to compute MACs on z_A and z_B

Active-Secure S2P from OT

- **Representation** of a secret bit x:

  A holds $x_A \in \{0,1\}$ B holds $x_B \in \{0,1\}$

  $$x = x_A \oplus x_B$$

- $xy = (x_A \oplus x_B)(y_A \oplus y_B) = x_A y_A \oplus x_B y_A \oplus x_A y_B \oplus x_B y_B$
- **And** secure computation of z=xy:

  A uses aAND to get a MAC on $t_A = x_A y_A$

  B uses aAND to get a MAC on $t_B = x_B y_B$

  Active-secure computation of $t = x_A y_A \oplus x_B y_B$

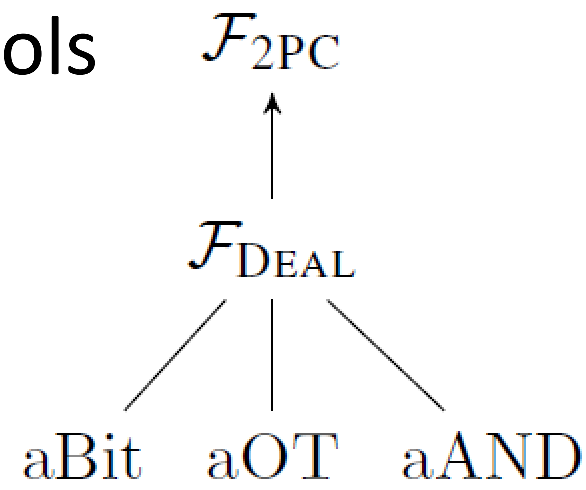
- They call aOT to securely compute

  $u = x_B y_A$ and $v = x_A y_B$

- Then they securely compute $z = t \oplus u \oplus v$

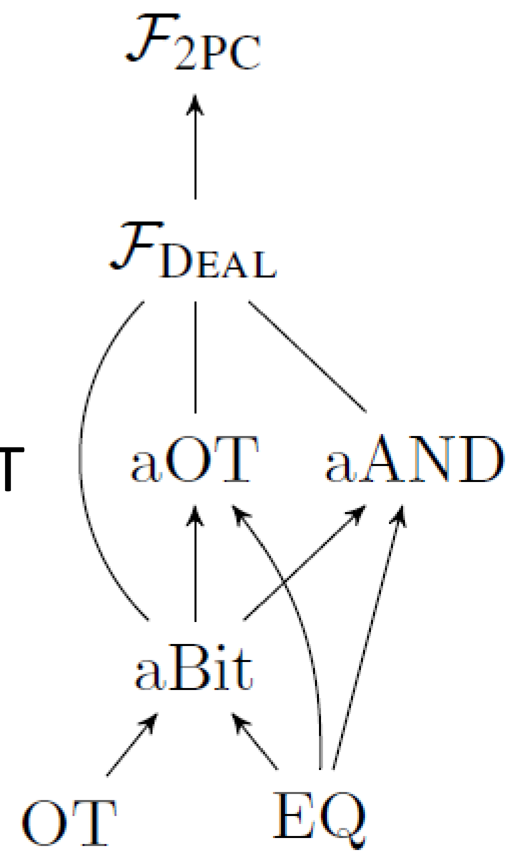
Overview of Protocol

- We implement a dealer functionality which serves a lot of random aBits, random aOTs and random aANDs
- Can be used to implement the non-random version of the primitives using simple random self-reducibility protocols like ROT→OT
- Can then implement secure 2PC as on the previous slides

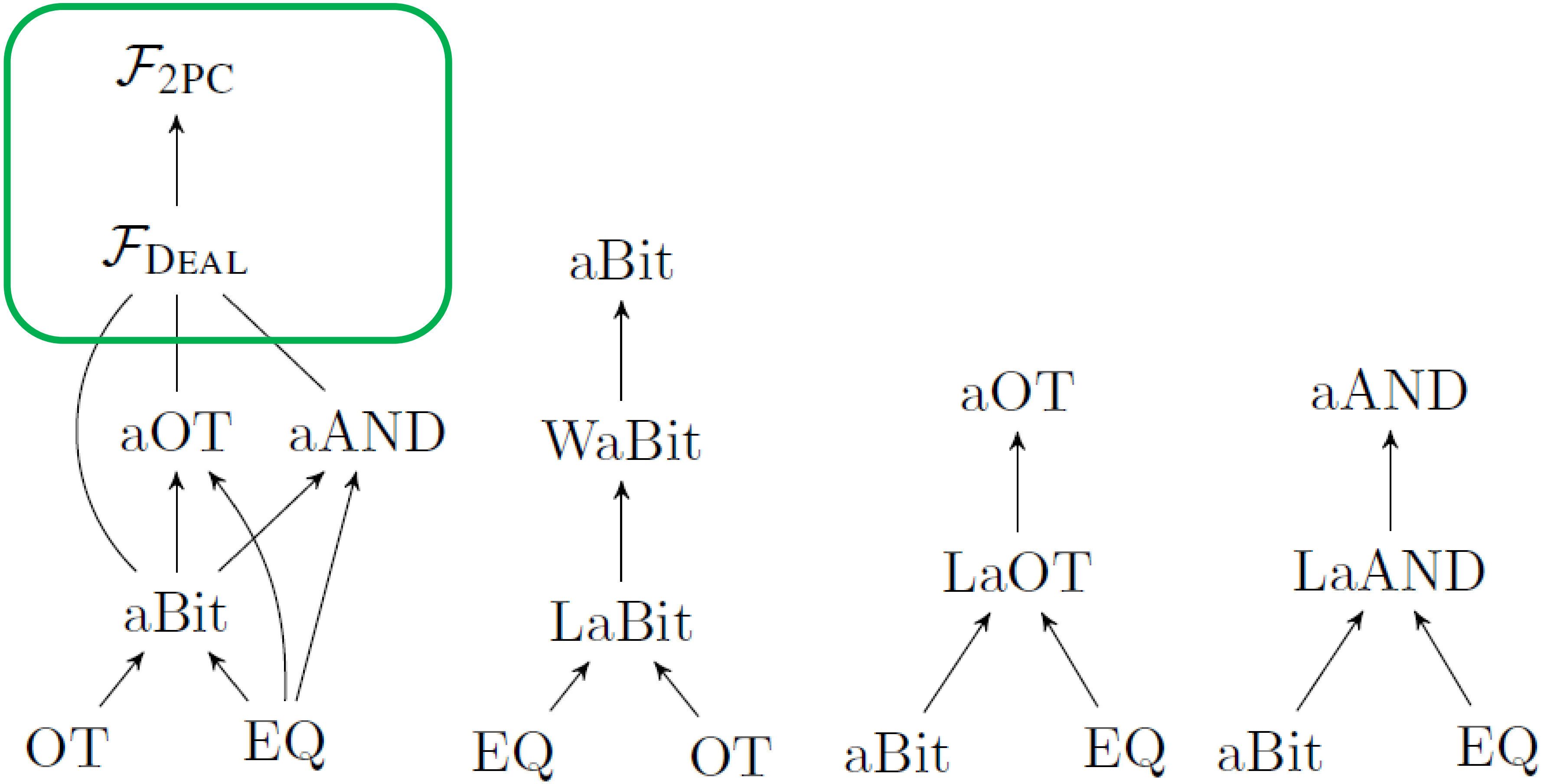


A Bit More Details

- We first use a few OTs + a pseudo-random generator and one secure equality check to implements a lot (any polynomial) number of random aBits
- We show how to turn a few aBits into one aOT
  - Uses one more EQ test overall and a few applications of a hash function H per aOT
- We show how to turn a few aBits into one aAND
  - Uses one more EQ test overall and a few applications of H per aOT

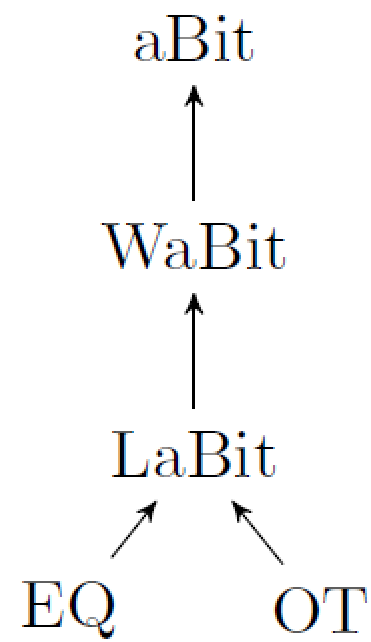


Even More Details



Random Authenticated Bits

- First we use a few OTs to generate a few aBits with very long keys
  - They are **L**eaky in that a few of the authenticated bits might leak to the key holder
- Then we turn our heads and suddenly have a lot of aBits with short keys
  - They are **W**eak in that a few bits of the global key might leak to the MAC holder
- Then we fix that problem using an extractor



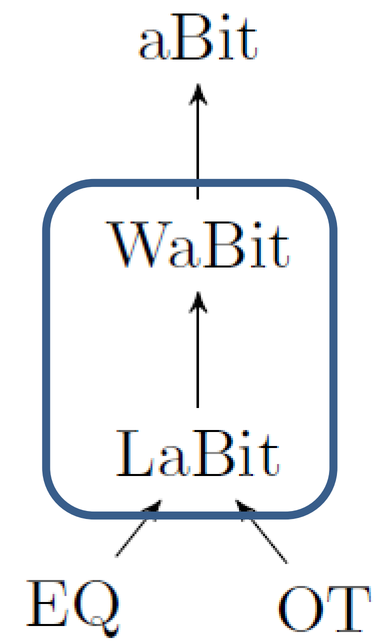
Turning Our Heads

$j=1, \dots, k$

- $N_j = L_j \oplus y_j\Gamma$ for Γ, L_j, $N_j \in\{0,1\}^n$
  - Think k = 160 and n = 1,000,000,000

$i=1, \dots, n$

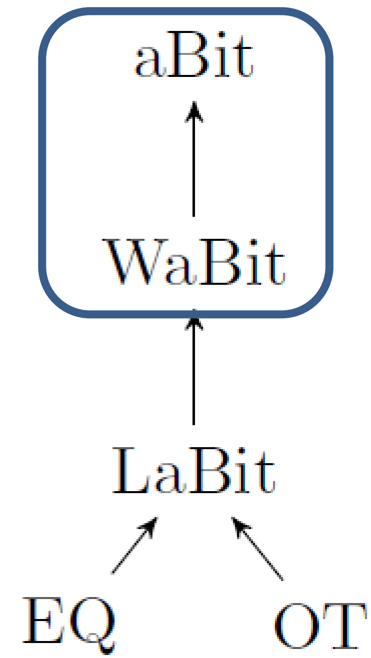
- Define $\Delta\in\{0,1\}^k$ and x_i and M_i, $K_i\in\{0,1\}^k$
- Global key to bits: $x_i = \Gamma_i$
- Bits to global key: $\Delta_j = y_j$
- MAC bits to key bits: $K_{ij} = N_{ji}$
- Key bits to MAC bits: $M_{ij} = L_{ji}$
- $N_{ji} = L_{ji} \oplus y_j\Gamma_i \Rightarrow K_{ij} = M_{ij} \oplus \Delta_j x_i$
  $\Rightarrow K_i = M_i \oplus \Delta x_i \Rightarrow M_i = K_i \oplus x_i\Delta$



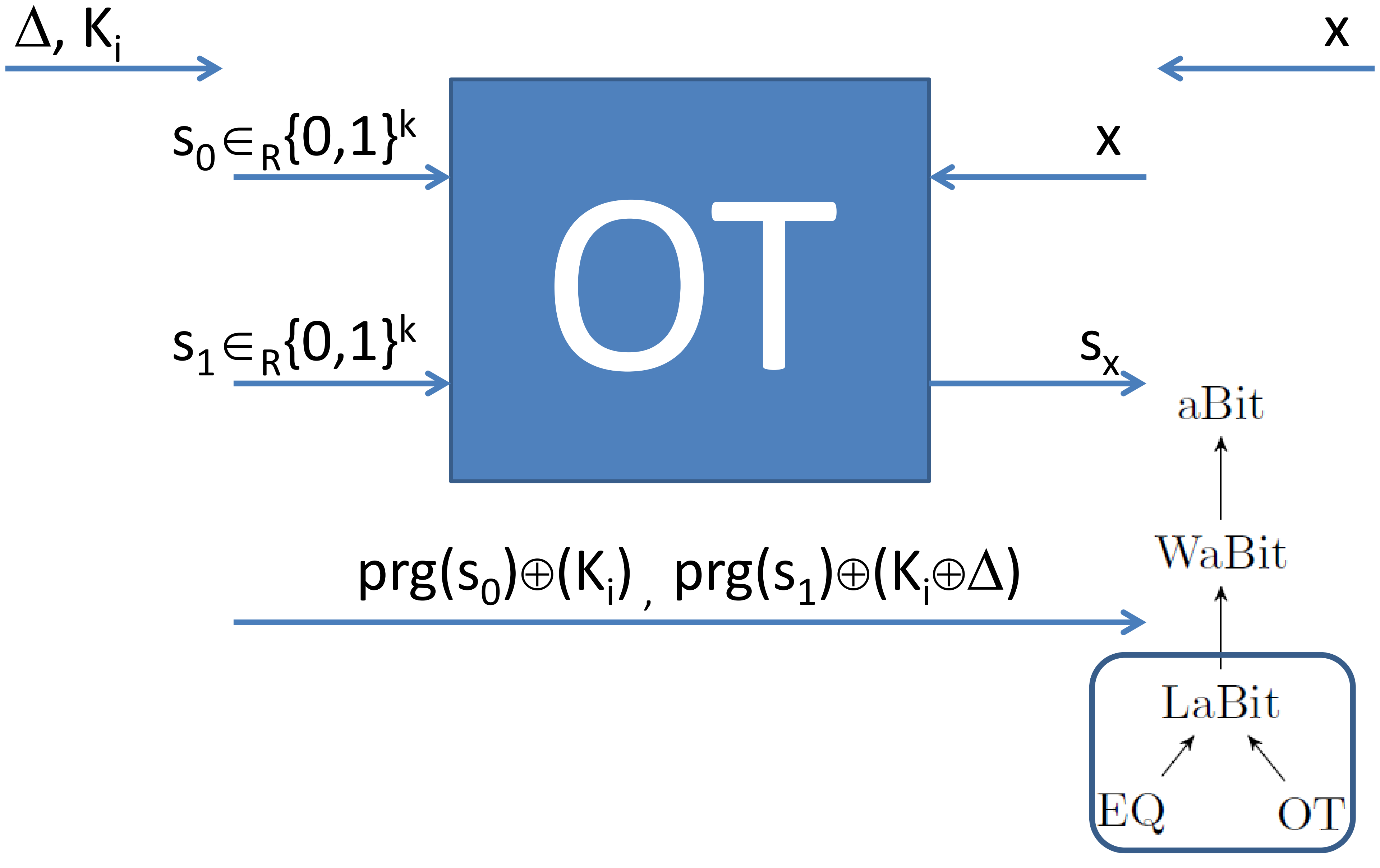
Extracting

$i = 1, \ldots, n$

- $M_i = K_i \oplus x_i\Delta$
  - A few bits of Δ are know to the adversary
- Owner of Δ picks a random matrix $X \in \{0,1\}^{k/2 \times k}$
- $\underline{M}_i = X\,M_i$ (in GF(2))
- $\underline{K}_i = X\,K_i$
- $\underline{\Delta} = X\,\Delta$
- $\underline{M}_i = X\,M_i = X(K_i \oplus x_i\Delta)$
  $= XK_i \oplus x_iX\Delta = \underline{K}_i \oplus x_i\underline{\Delta}$
- So still correct and now secure as a random matrix is a good extractor



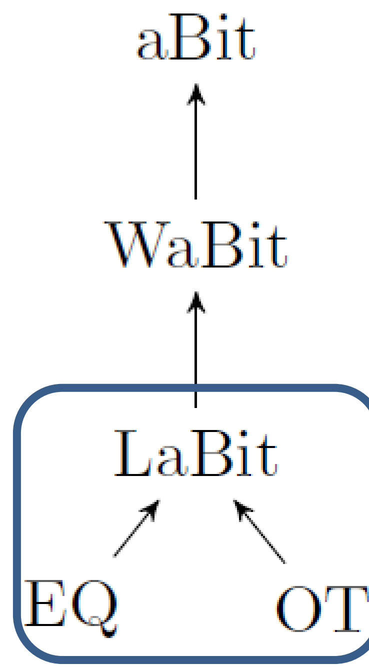
OT → aBit



Δ, K_i
x
s_0 ∈_R {0,1}^k
x
OT
s_1 ∈_R {0,1}^k
s_x
aBit
WaBit
prg(s_0)⊕(K_i) , prg(s_1)⊕(K_i⊕Δ)
LaBit
EQ
OT

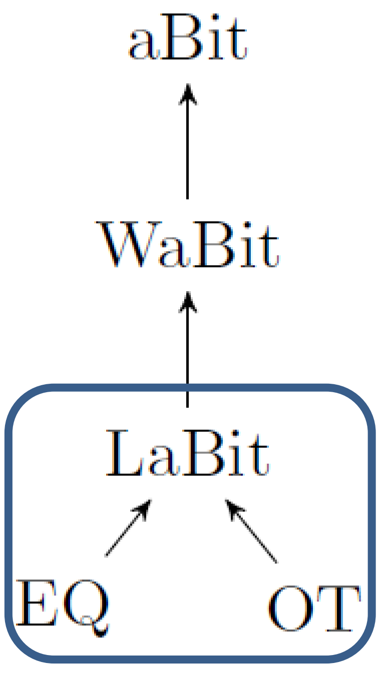
Problem 1 and Hint of the Fix

- Last problem is that Alice might not use the same Δ in all k implementations of aBits from OTs
- Is handled by implementing twice as many aBits as needed and then doing cut-and-choose in which we check that half of them were done with the same Δ
  - Needs a small trick to avoid revealing the value of Δ

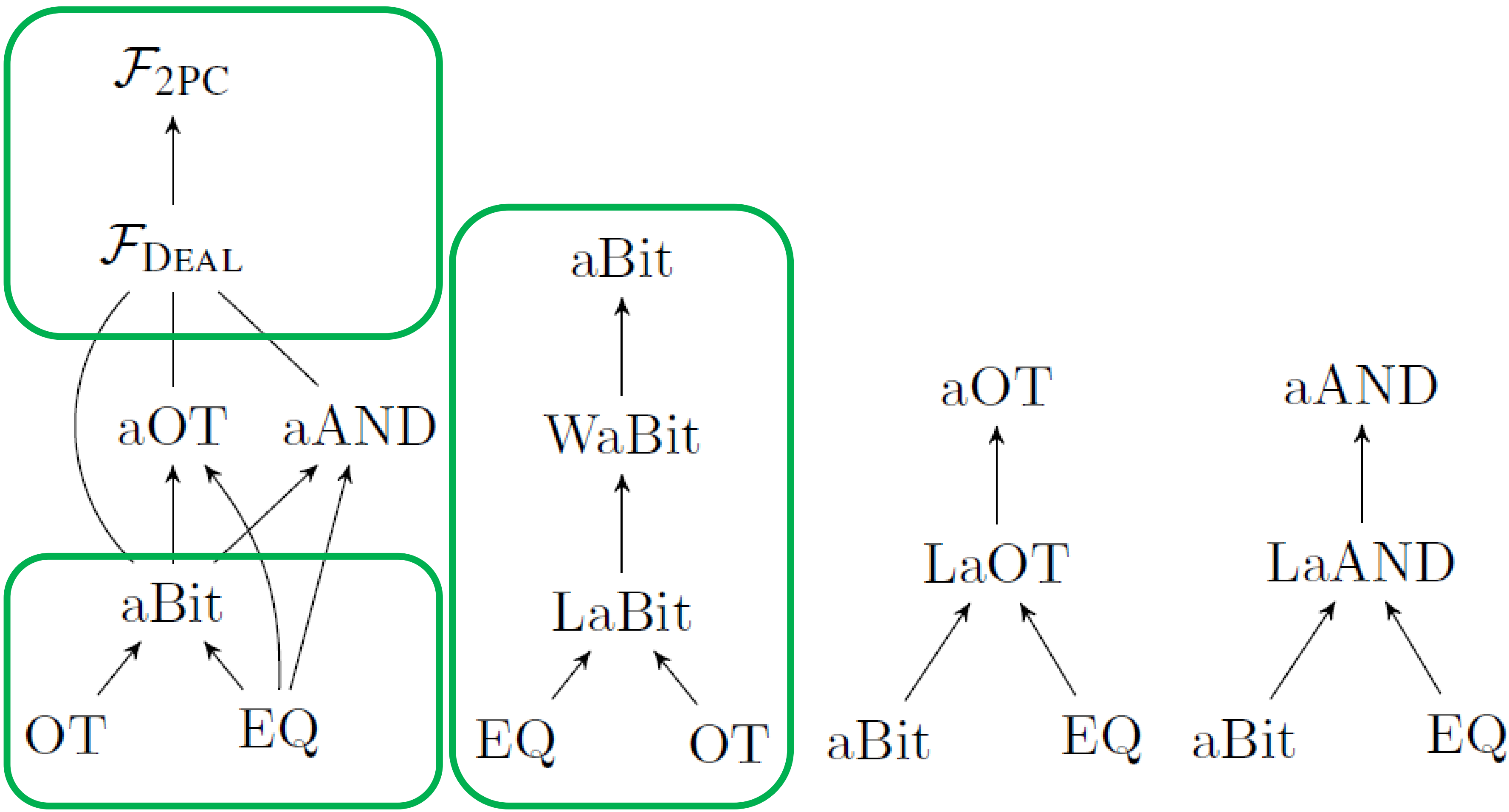


Problem 2 and Hint of the Fix

- The cut-and-choose stills lets Alice use a different Δ in a few of the aBits
- Can, however, show that a different Δ in a given aBit is no worse than letting Alice learn the bit being authenticatec in that aBit
  - An information theoretic simulation argument
- This leaves us with a few aBits of which a few bits have leaked to Alice

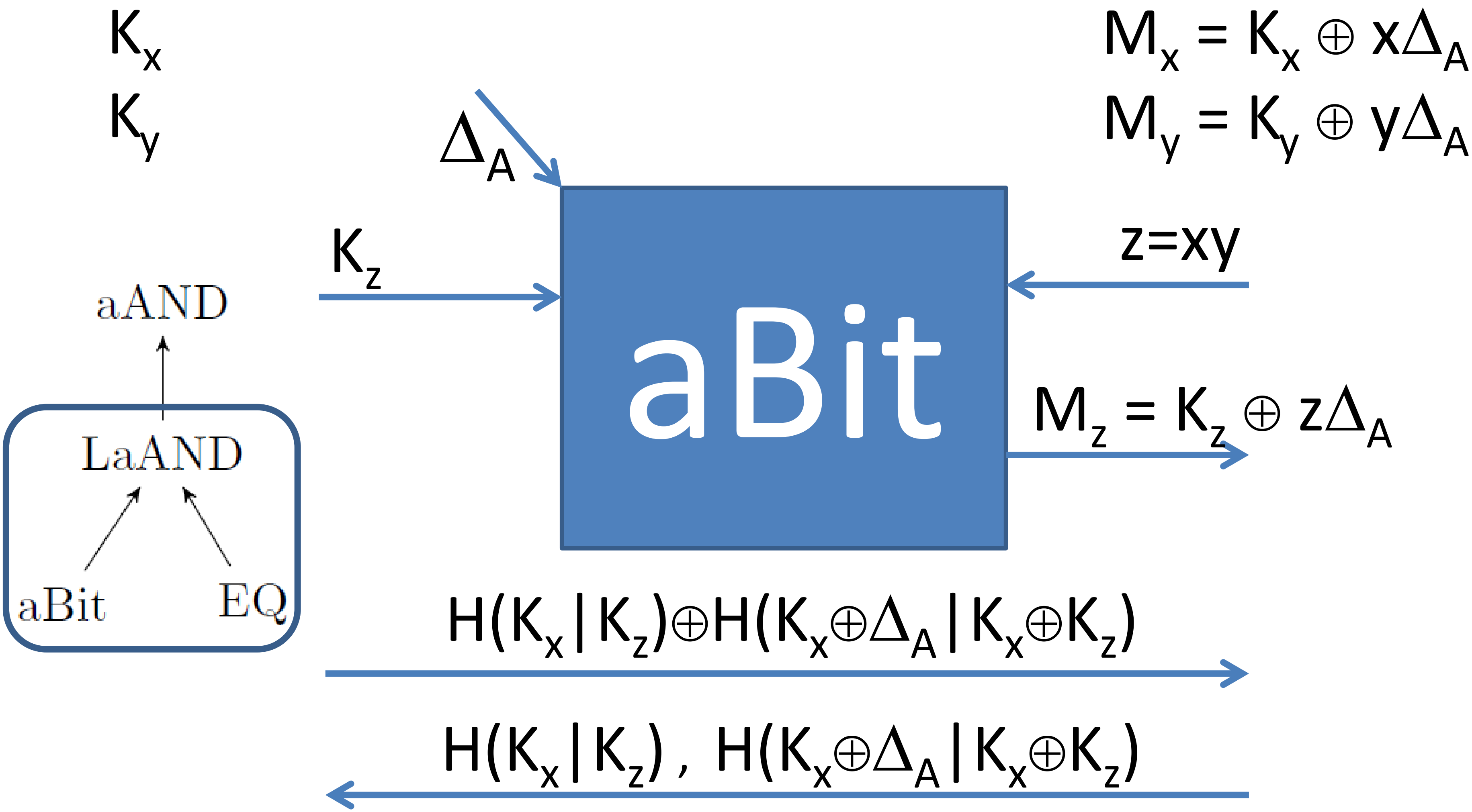


Status



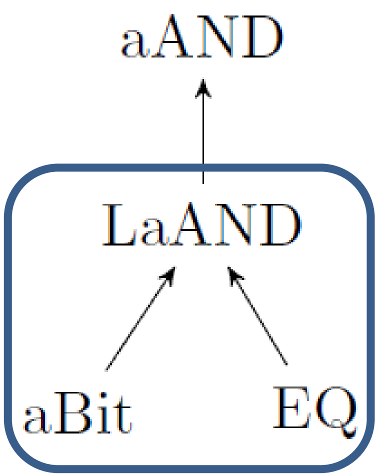
$\mathcal{F}_{2PC}$
$\mathcal{F}_{\text{DEAL}}$
aOT
aAND
aBit
OT
EQ
aBit
WaBit
LaBit
EQ
OT
aOT
LaOT
aBit
EQ
aAND
LaAND
aBit
EQ

Authenticated AND

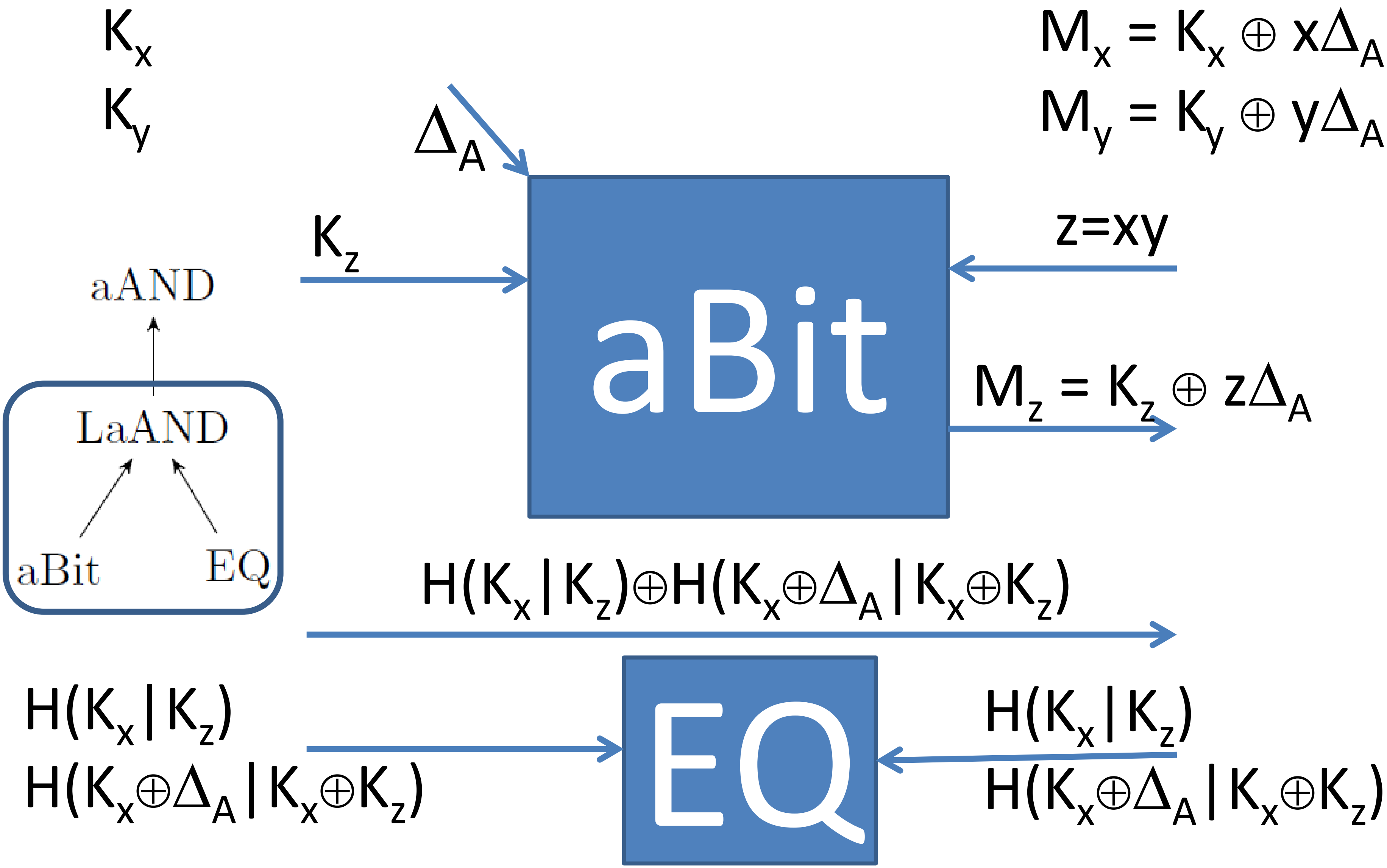


Problem and a Fix

- If Alice sends an incorrect value, then the response of Bob depends on x
- Instead we do a secure comparison of the response and what Alice expects
- If Alice sends an incorrect value, then the response of Bob depends on x
- So, a cheating Alice will fail to give the right input to the comparison with some constant probability
- So, Alice can learn x in O(k) of the aANDs with probability at most 2^{-k}

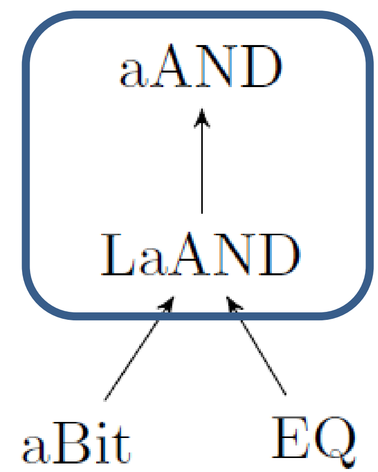


Authenticated AND



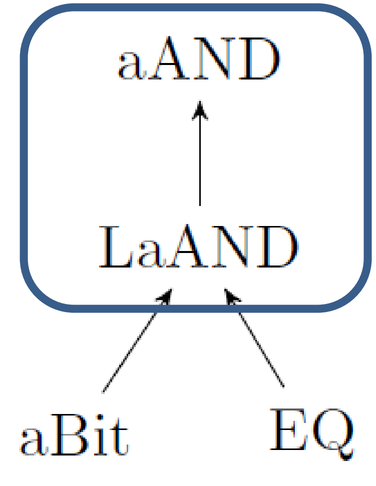
Combining

- Generate Bn LaAnds (x_i, y_i, z_i)
- Bob divides them randomly into n buckets of size B, where all triples in the same bucket have the same y-value
- For each bucket (x_1,y,z_1), …, (x_B,y,z_B), securely compute $x = x_1 \oplus \ldots \oplus x_B$
  $z = z_1 \oplus \ldots \oplus z_B$
  and output (x,y,z)
- Correctness: $xy = (x_1 \oplus \ldots \oplus x_n)y = x_1y \oplus \ldots \oplus x_ny = z_1 \oplus \ldots \oplus z_n = z$
- Secure if just one x_i was not leaked



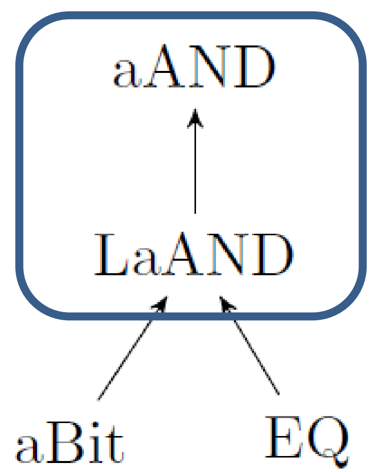
Problem and a Fix

- If Bob puts triples with different y-values in a bucket the correctness breaks
- He uses his MACs on the y-values to prove that they are the same
- Specifically he sends $x_1 \oplus x_2$, $x_2 \oplus x_3$, …, $x_{B-1} \oplus x_B$
- Alice checks that they are all 0
- Bob sends along the MACs of the Xors to prove correctness, which is possible by the Xor-homomorphism

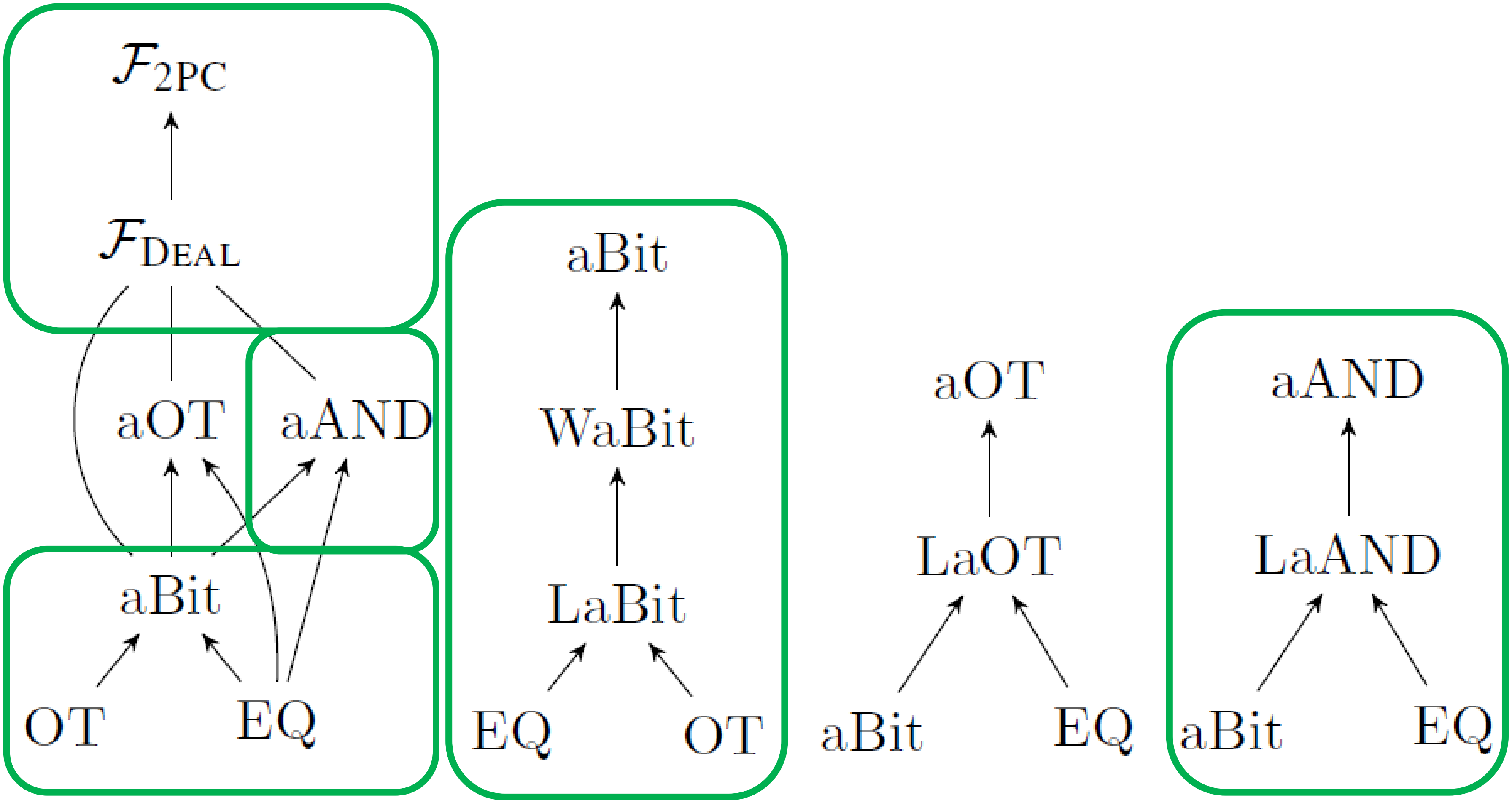


Security

- Probability that there exists a bucket where all triples are leaky can be upper bounded by
$$(2n)^{-(B-1)} = 2^{-(1+\log(n))(B-1)}$$
- In particular, for a fixed overhead B, the security increases with n, the number of gates we have to handle
- Example: $B=4$ and $n=1,000,000$ gives security around 2^{-63}
- Our implementation uses a fixed $B=4$ as we do massive computations

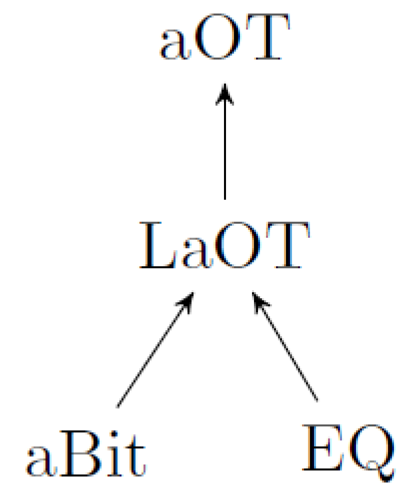


Status



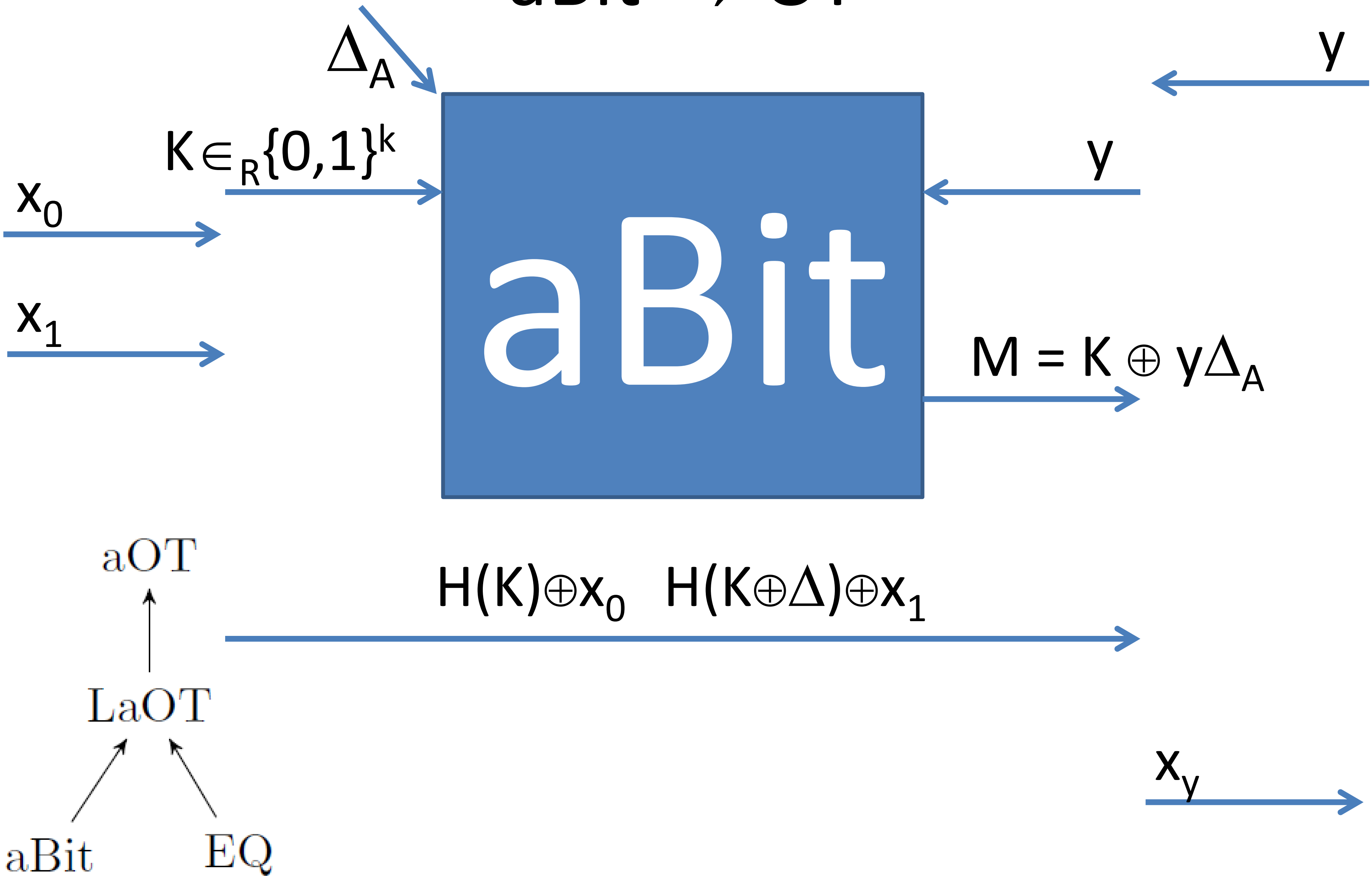
$\mathcal{F}_{\text{2PC}}$
$\mathcal{F}_{\text{DEAL}}$
aOT
aAND
aBit
OT
EQ
aBit
WaBit
LaBit
EQ
OT
aOT
LaOT
aBit
EQ
aAND
LaAND
aBit
EQ

Authenticated OT

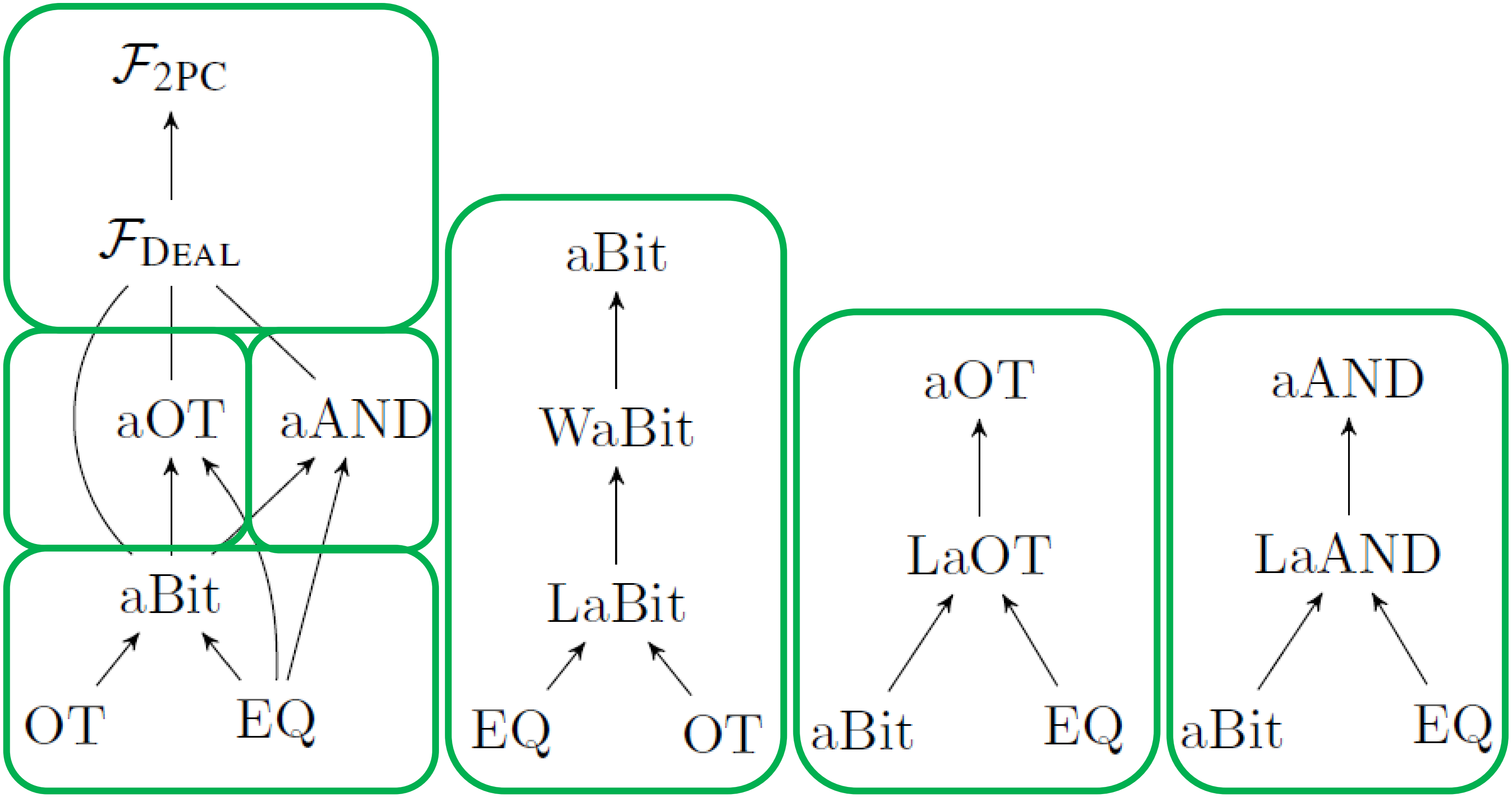


- Same, same, …
- First the parties run an OT
- Then they use runs of aBit to get their inputs and outputs authenticated
- Then they do a slightly more involved version of the Xor-of-hash challenge-response technique
- Then we combine to get rid of a few leaked bits
- Only problem is that we actually did not implement OT efficiently yet

aBit $\rightarrow$ OT



Status



$\mathcal{F}_{2PC}$
$\mathcal{F}_{\text{DEAL}}$
aOT
aAND
aBit
OT
EQ
aBit
WaBit
LaBit
EQ
OT
aOT
LaOT
aBit
EQ
aAND
LaAND
aBit
EQ

Benchmarking

- We implemented the protocol in Java and ran it between two different machines on the intranet of Aarhus university
- We did secure encryption using AES
  - Key is Xor shared between the parties
  - Plaintext is input by Alice
  - Both parties learn the ciphertext
- Circuit of AES is about 34000 gates

- ℓ: Number of 128-bit blocks encrypted
- G: # of gates
- σ: Statistical security level
  - a bucket is bad with probability $2^{-\sigma}$
- T_{pre}: Seconds for implementing Dealer
  - Can be done before inputs arrive
- T_{onl}: Time spend evaluating once random values are dealt
- $T_{tot} = T_{pre} + T_{onl}$

ℓ	G	σ	T_{pre}	T_{onl}	T_{tot}/ℓ	G/T_{tot}
1	34,520	55	38	4	44	822
27	922,056	55	38	5	1.6	21,545
54	1,842,728	58	79	6	1.6	21,623
81	2,765,400	60	126	10	1.7	20,405
108	3,721,208	61	170	12	1.7	20,541
135	4,642,880	62	210	15	1.7	20,637

ℓ	G	σ	T_{pre}	T_{onl}	T_{tot}/ℓ	G/T_{tot}
256	8,739,200	65	406	16	1.7	20,709
512	17,478,016	68	907	26	1.8	18,733
1,024	34,955,648	71	2,303	52	2.3	14,843
2,048	69,910,912	74	5,324	143	2.7	12,788
4,096	139,821,440	77	11,238	194	2.8	12,231
8,192	279,642,496	80	22,720	258	2.8	12,170
16,384	559,284,608	83	46,584	517	2.9	11,874