UNIVERSITÄT BERN

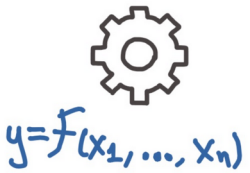
Secure Multiparty Computation Based on Linear Secret Sharing

Orestis Alpos
University of Bern

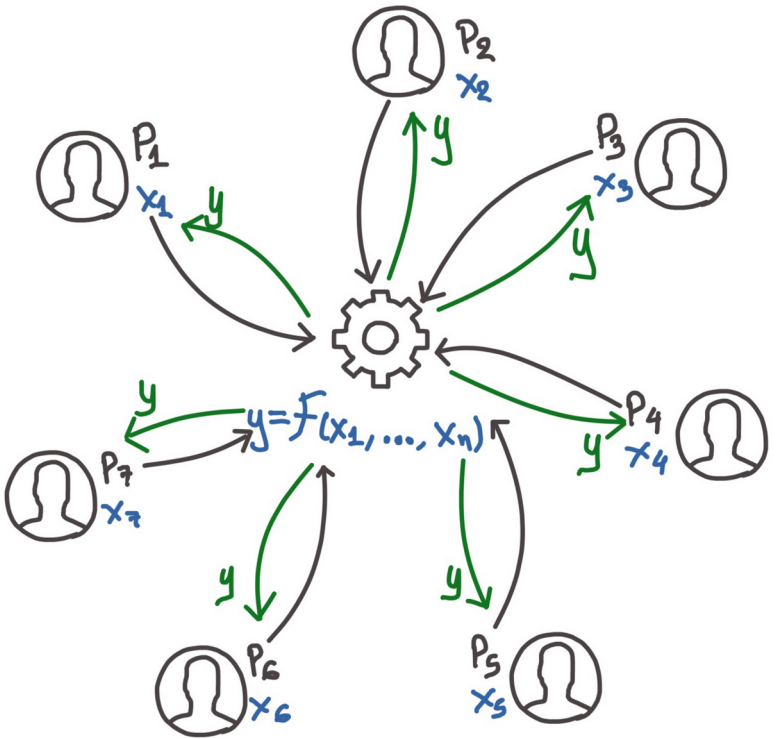
CRYPTO group, internal research seminar
November 2020

What is MPC

What is MPC



What is MPC



Properties

- Privacy: Any information learnt by P_i can be derived by x_i and y
  - Auction, output y is the highest bid
  - Nothing should be learnt for the other bids
  - But we do learn that they are lower than
- Correctness: The output received by each player is correct
  - Auction, party with highest bid will win
  - All parties will know it

Properties

- Independence of inputs: Corrupt parties must choose inputs independent of honest parties
- Fairness: Corrupt parties receive output if and only if honest parties do
- Guaranteed output delivery (Robustness): Corrupt parties cannot prevent honest parties from receiving the output
  – Stronger than fairness

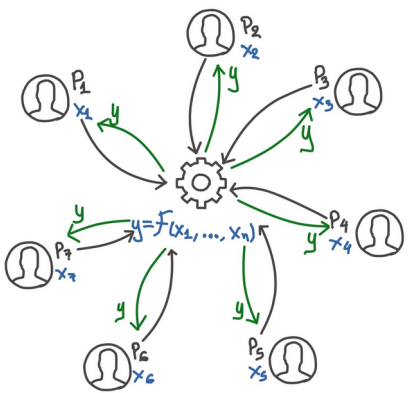
Not exhaustive
Not all properties always guaranteed

Formal definition

UNIVERSITÄT BERN

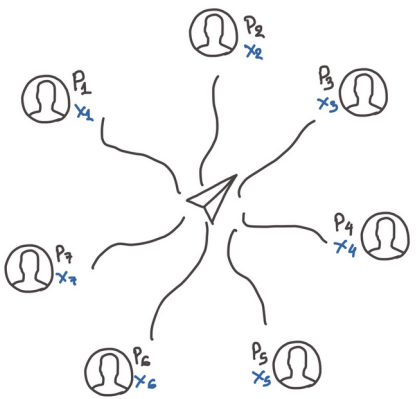
Ideal world

- An external trusted functionality does the computation
- Properties hold by definition



Real world

- No trusted party, parties run protocol
- Prove that the adversary cannot do any worse than in ideal world



Additional definition parameters

- Adversarial behavior
  - Passive (honest-but-curious, semi-honest)
  - Active (malicious)
  - Covert
- Corruption strategy
  - Static
  - Adaptive
  - Mobile (proactive security)
- Corruption thresholds
  - Honest supermajority ($t < n / 3$)
  - Honest majority ($t < n / 2$)
  - Dishonest majority ($t < n$)
- Type of security
  - Information theoretic
  - Computational
- Modular composition
  - Sequential (stand-alone setting)
  - Parallel (universal composability, UC)

Security with abort

MPC approaches

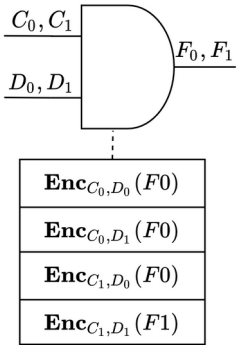
First step

- Write *F* as an arithmetic circuit *C* of *add* and *multiply* gates.
- Evaluate *C* gate by gate
- Addition and multiplication are universal over F_p

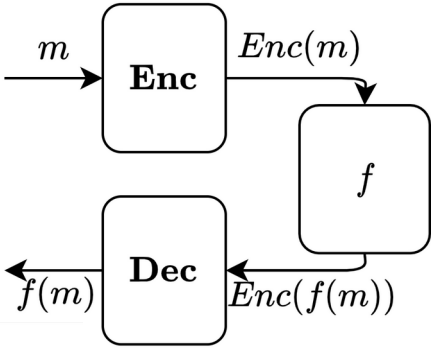
Whatever needs to be computed,
can be computed securely

Three approaches to evaluate the circuit

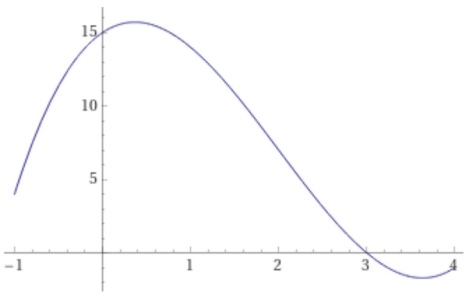
- Garbled circuits



- Homomorphic encryption

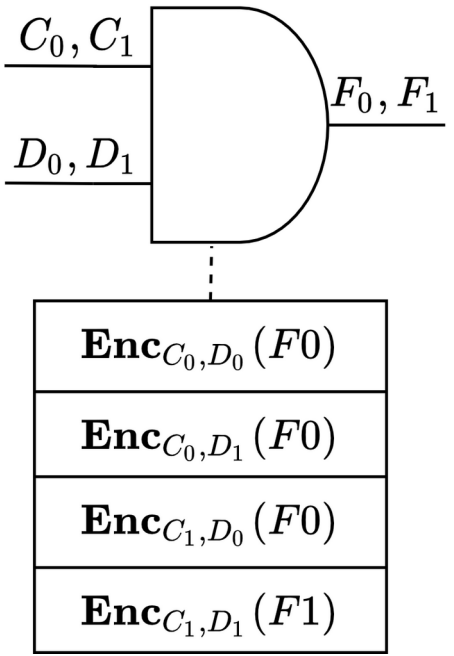


- Secret sharing

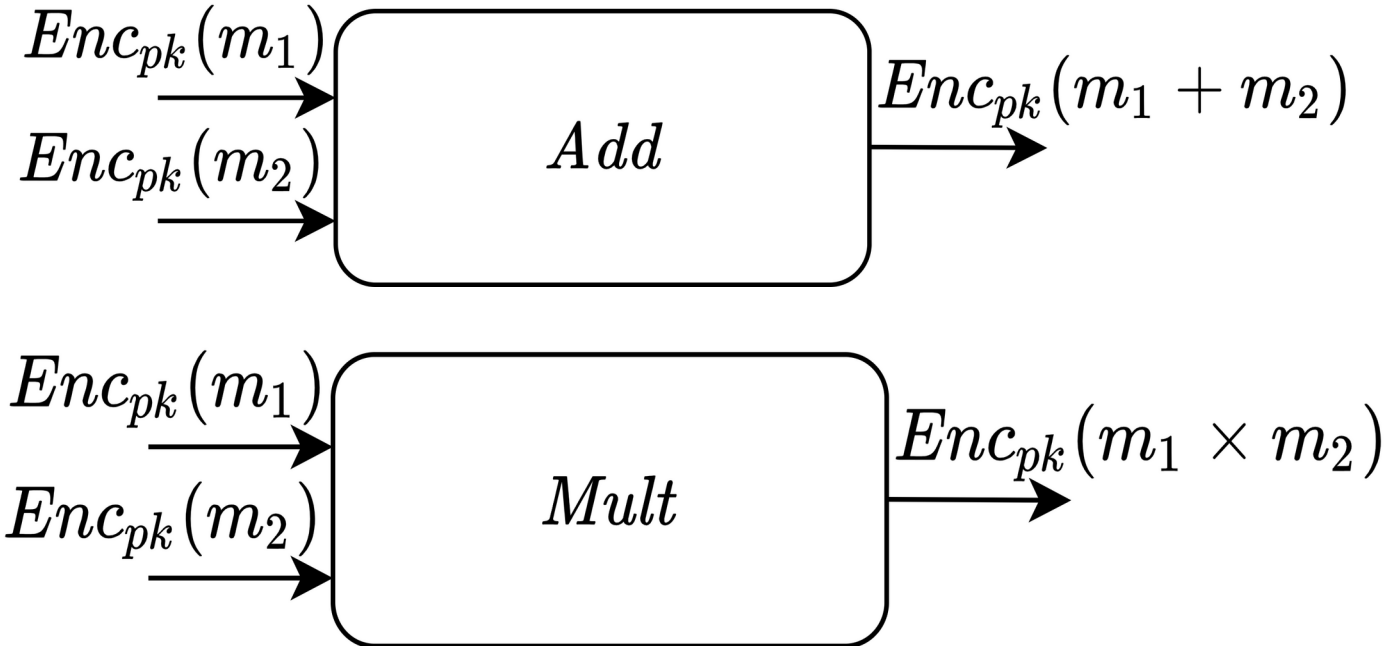


1. Garbled circuits

- Garbler and Evaluator [Yao82]
- Treat gate as matrix
  For example, AND gate has 4 rows, one for each possible input pair
- Encrypt each row, send only the keys that decrypt one input
- When output also encrypted, we can use it as input to the next gate



2. Fully homomorphic encryption

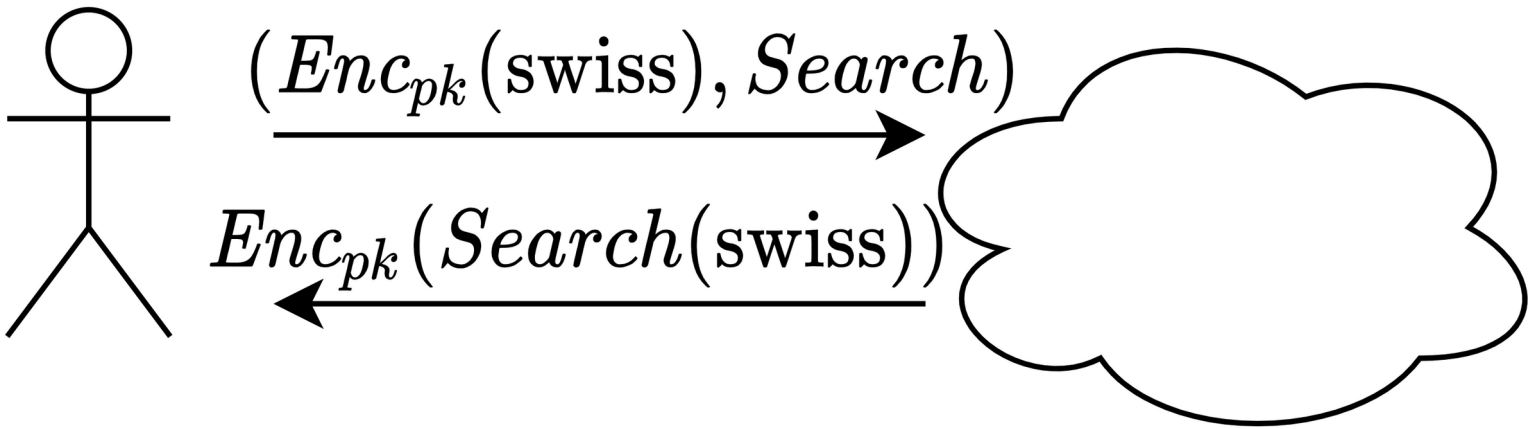


- *Add* and *Mult* are specific to the scheme

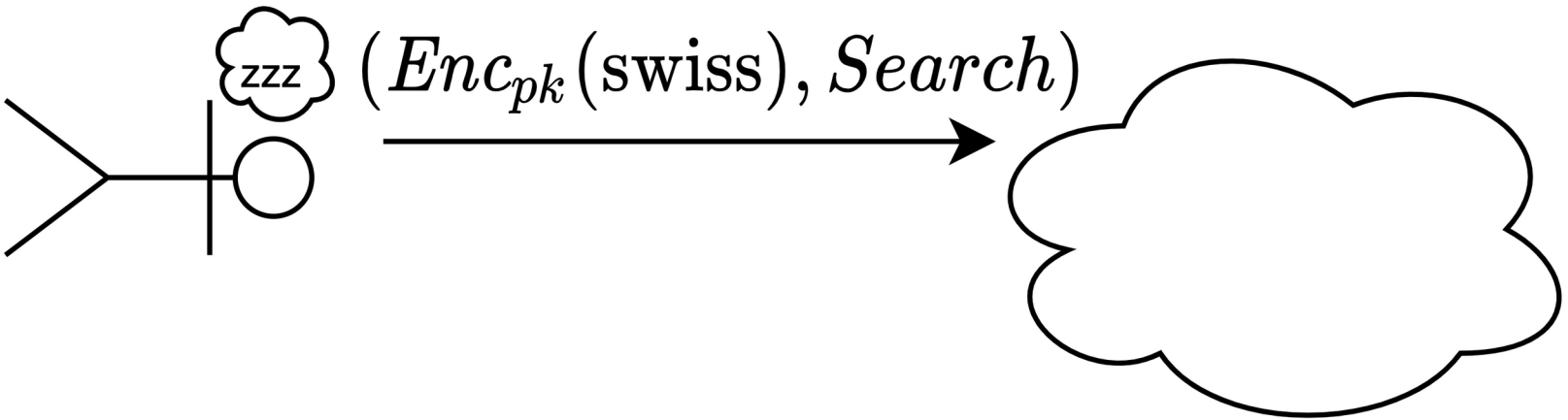
2. Fully homomorphic encryption

- For MPC, we also need partial decryption (*sk* is shared among parties)
- For passive, computational security with two rounds of communication:
- Each p_i encrypts its input and broadcasts
- Parties compute the circuit on ciphertexts
- Each p_i partially decrypts result and broadcasts
- Parties combine partial decryptions to obtain result

2. Fully homomorphic encryption is promising



2. Fully homomorphic encryption is slow



2. Fully homomorphic encryption vs (P/S) HE

- Partially homomorphic encryption
- Somewhat homomorphic encryption
- Examples:
  - ElGamal: $Enc_Y(m) = (g^r, mY^r)$
  - RSA: $Enc_e(m) = m^e$
  - Both partially homomorphic under multiplication

2. An example from SPDZ

UNIVERSITÄT BERN

- Use SHE to obtain an additive sharing of number *m* [DPSZ12]

Protocol Reshare

Usage: Input is $e_{\mathbf{m}}$, where $e_{\mathbf{m}} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$ is a public ciphertext and a parameter enc, where $enc =$ NewCiphertext or $enc =$ NoNewCiphertext. Output is a share $\mathbf{m}_i$ of $\mathbf{m}$ to each player P_i; and if $enc =$ NewCiphertext, a ciphertext $e'_{\mathbf{m}}$. The idea is that $e_{\mathbf{m}}$ could be a product of two ciphertexts, which Reshare converts to a "fresh" ciphertext $e'_{\mathbf{m}}$. Since Reshare uses distributed decryption (that may return an incorrect result), it is not guaranteed that $e_{\mathbf{m}}$ and $e'_{\mathbf{m}}$ contain the same value, but it *is* guaranteed that $\sum_i \mathbf{m}_i$ is the value contained in $e'_{\mathbf{m}}$.

Reshare$(e_{\mathbf{m}}, enc)$:

1. Each player P_i samples a uniform $\mathbf{f}_i \in (\mathbb{F}_{p^k})^s$. Define $\mathbf{f} := \sum_{i=1}^{n} \mathbf{f}_i$.
2. Each player P_i computes and broadcasts $e_{\mathbf{f}_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{f}_i)$.
3. Each player P_i runs Π_{ZKPoPK} acting as a prover on $e_{\mathbf{f}_i}$. The protocol aborts if any proof fails.
4. The players compute $e_{\mathbf{f}} \leftarrow e_{\mathbf{f}_1} \boxplus \cdots \boxplus e_{\mathbf{f}_n}$, and $e_{\mathbf{m}+\mathbf{f}} \leftarrow e_{\mathbf{m}} \boxplus e_{\mathbf{f}}$.
5. The players invoke $\mathcal{F}_{\text{KeyGenDec}}$ to decrypt $e_{\mathbf{m}+\mathbf{f}}$ and thereby obtain $\mathbf{m} + \mathbf{f}$.
6. P_1 sets $\mathbf{m}_1 \leftarrow \mathbf{m} + \mathbf{f} - \mathbf{f}_1$, and each player P_i $(i \neq 1)$ sets $\mathbf{m}_i \leftarrow -\mathbf{f}_i$.
7. If $enc =$ NewCiphertext, all players set $e'_{\mathbf{m}} \leftarrow \mathsf{Enc}_{pk}(\mathbf{m} + \mathbf{f}) \boxminus e_{\mathbf{f}_1} \boxminus \cdots \boxminus e_{\mathbf{f}_n}$, where a default value for the randomness is used when computing $\mathsf{Enc}_{pk}(\mathbf{m} + \mathbf{f})$.

Fig. 4. The sub-protocol for additively secret sharing a plaintext $\mathbf{m} \in (\mathbb{F}_{p^k})^s$ on input a ciphertext $e_{\mathbf{m}} = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{m})$.

3. Secret sharing

UNIVERSITÄT BERN

- Share a value *x* among *n* participants, so that [Shamir79]
  - *t* + *1* can recover the secret
  - any *t* have no information about it

- Share
  - Degree-*t* random polynomial: $f(x) = k + a_1 x + \ldots + a_t x^t$
  - Give each party the share $s_i = f(i)$

- Reconstruct
  - *t* + 1 pairs (i, s_i) uniquely determine *f*
  - Lagrange interpolation

3. General secret sharing (LSSS)

- Share a value *x* among *n* parties, given access structure *A*, so that [CDM00]
  - An authorized set in *A* can recover the secret
  - Any other set has no information about it

- MSP (labeled 2D matrix *M*) is equivalent to LSSS

- Share
  - Random vector $\boldsymbol{r} = (k, a_1, \ldots, a_{d-1})$
  - Calculate shares as $\boldsymbol{s} = M\boldsymbol{r}$

- Reconstruct
  - For quorum *A* with shares $\boldsymbol{s}_A$ find recombination vector $\boldsymbol{\lambda}_A$ such that $\boldsymbol{\lambda}_A M_A = \boldsymbol{e}$
  - The value is $x = \boldsymbol{\lambda}_A \boldsymbol{s}_A$

3. Secret sharing - Add

- Players hold sharings
  - *[x]* of *x*, made with *deg-t* polynomial
  - *[y]* of y, made with *deg-t* polynomial

- Obtain sharing *[x + y]* of *x + y* by locally adding shares

- No interaction

3. Secret sharing - Multiply

- Players hold sharings
  - *[x]* of *x*, made with *deg-t* polynomial f_1
  - *[y]* of y, made with *deg-t* polynomial f_2

- Obtain sharing *[xy]* of *xy* by locally multiplying shares

- But polynomial $g = f_1 f_2$ has degree *2t*

3. Secret sharing - Multiply

- Degree reduction
- Luckily, we have *2t + 1* shares of *g* (we started with *t < n / 2*)
- These shares determine *g(0)* as $g(0) = \sum_{i=1}^{2t+1} \lambda_i g(i)$
- Each p_i shares *g(i)* with deg-t polynomial
- Parties now calculate $[g(0)] = \sum_{i=1}^{2t+1} \lambda_i [g(i)]$
- This is a sharing of *g(0)* with the correct degree

3. Secret sharing - Multiply

- Similar idea for LSSS (Maurer)
- Requires the exchange of n^2 elements (each party send n elements)

3. Secret sharing - Multiply with Beaver trick

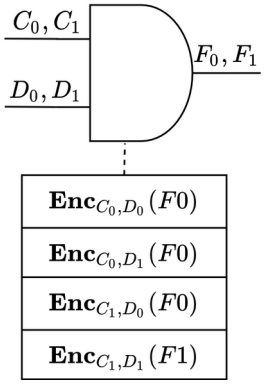
UNIVERSITÄT BERN

- Assume *[a], [b], [c]*, with *ab* = *c* and *a,b,c* unknown, are available [Beaver91]
- Parties open *[ε]* = *[x]* – *[a]*. Reconstruct *ε*
- Parties open *[δ]* = *[y]* – *[b].* Reconstruct *δ*
- Parties compute *[z]* = *[c]* + *ε[b]* + *δ[a]* + *εδ* locally
- Now *2n* elements are exchanged (each party send *2* elements)

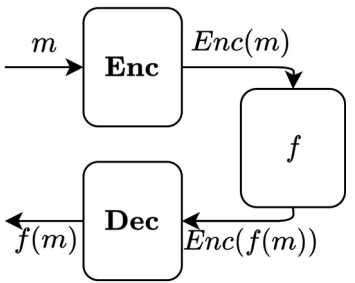
Three approaches to evaluate the circuit - Summary

UNIVERSITÄT BERN

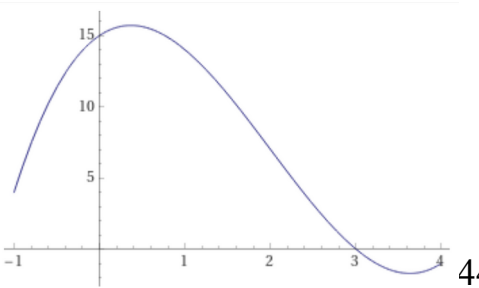
- Garbled circuits
  - 2PC
  - Low communication complexity
  - Practical and efficient for Boolean operations
  - Large circuit size for arithmetic operations



- Homomorphic encryption
  - Low communication complexity
  - Slow (computationally expensive operations)



- Secret sharing
  - No computationally expensive PK operations
  - High communication complexity
  - Number of rounds depends on multiplicative depth



Combine the three approaches: The preprocessing model

UNIVERSITÄT BERN

[DPSZ12]

- Very fast online phase
  - Information theoretic primitives
  - No PK
  - Assume everything is given

- We saw how parties can add and multiply values, given sharings + Beaver triples

- Slow offline phase
  - Create everything for online phase
  - Heavy HE
  - Does not depend on circuit *C*
  - (it is not really offline)

- We saw how parties can create sharings (Beaver triples is similar)

From passive to active security

From passive to active security

- Adversary can send false shares
- We need a way to verify
- One solution: Verifiable secret sharing (Commitments)
  - Information-theoretic
  - Computational

Don’t slow me down!

From passive to active security

- Sacrifice security properties to gain efficiency
- Dishonest majority, security with abort
- We can detect cheating, not correct it

Message Authentication Codes (MACs)

- Key $K=(\alpha, \beta_i)$, or $K=\alpha$ in SPDZ [DPSZ12]
- α is global but unknown to anyone. Assume we have a sharing $[\alpha]$
- Authenticate value a by $MAC_K(a) = \alpha a$
- For a shared value x, store $[x]$ and $[\alpha x]$
- Add $x + y$: $[x] + [y]$, $[\alpha x] + [\alpha y]$
- Multiply xy: $[xy]$, $[\alpha x][y]$ (using passive multiplication protocol twice)
- Output phase: Commit to the values, open α, check MACs

Thank you!

Orestis Alpos
orestis.alpos@inf.unibe.ch
crypto.unibe.ch/oa/
Twitter: @alpenliebious

UNIVERSITÄT BERN

[Yao82] DBLP:conf/focs/Yao82b
[Beaver91] DBLP:conf/crypto/Beaver91a
[CDM00] DBLP:conf/eurocrypt/CramerDM00
[DPSZ12] DBLP:conf/crypto/DamgardPSZ12