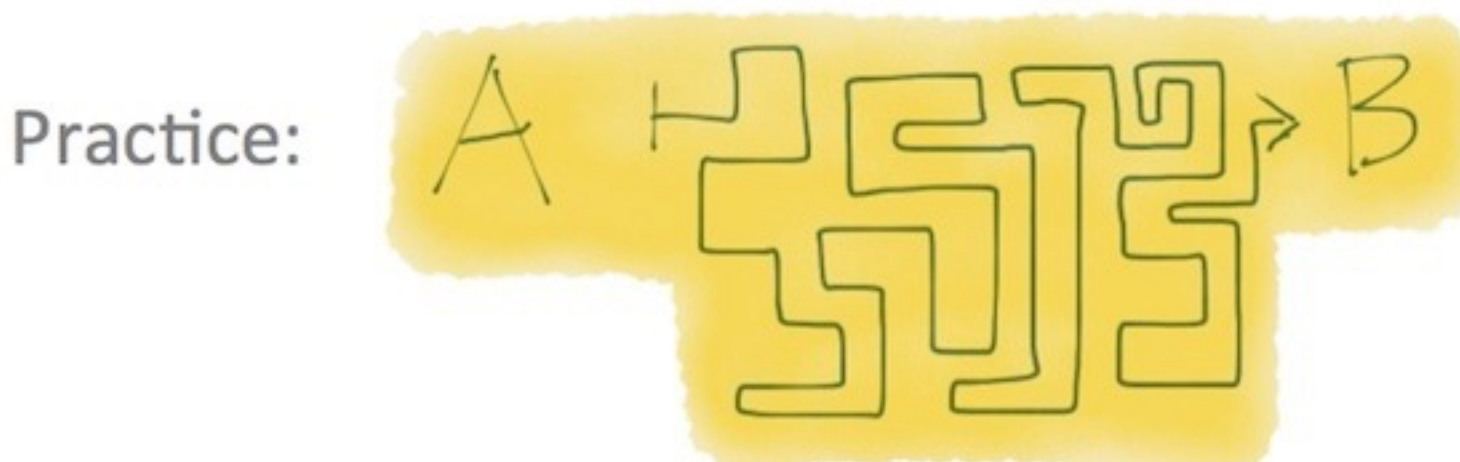
More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries

Gilad Asharov — **Hebrew University**
Yehuda Lindell — Bar-Ilan University
Thomas Schneider — Darmstadt
Michael Zohner — Darmstadt

EUROCRYPT 2015

From Theory to Practice

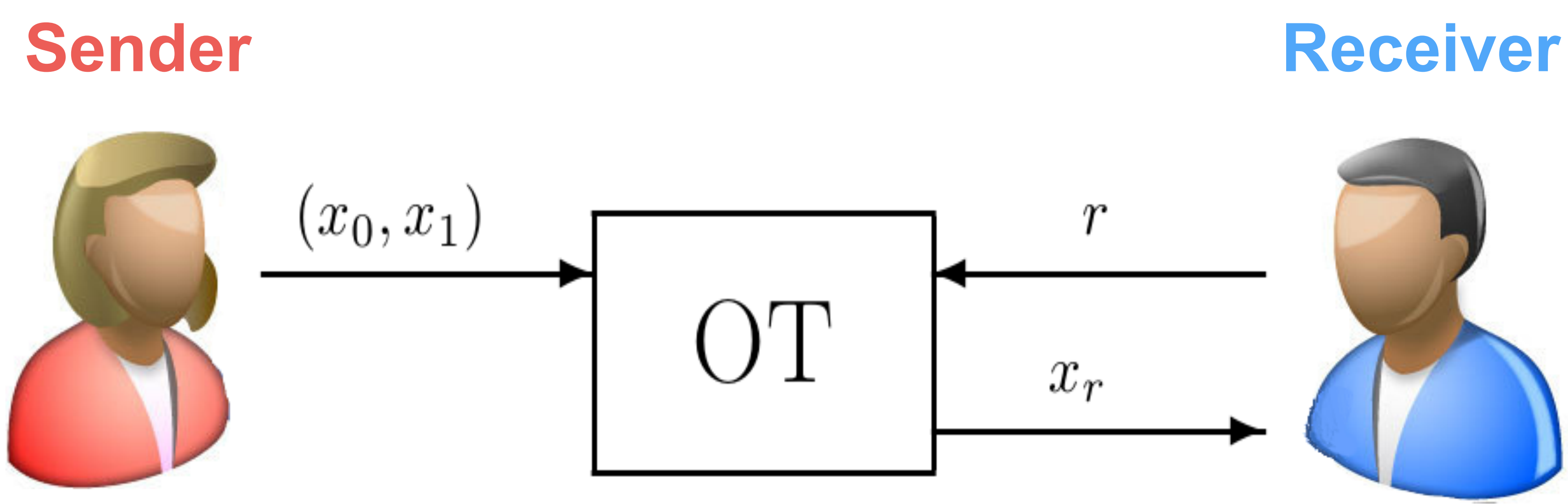
[Yao82,Yao86,GMW87,BGW88,CCD88,RB89,…]



Secure computation becomes practical!

[MNPS04,LP07,LPS08,PSSW09,KSS12,FN13,SS13,LR14,HKK+14, FJN14,NNOB12,LOS14,DZ13,DLT14,DCW13,JKO13]

1-out-of-2 Oblivious Transfer



- **INPUT: Sender** holds two strings (x_0,x_1), **Receiver** holds r
- **OUTPUT: Sender** learns nothing, **Receiver** learns x_r,

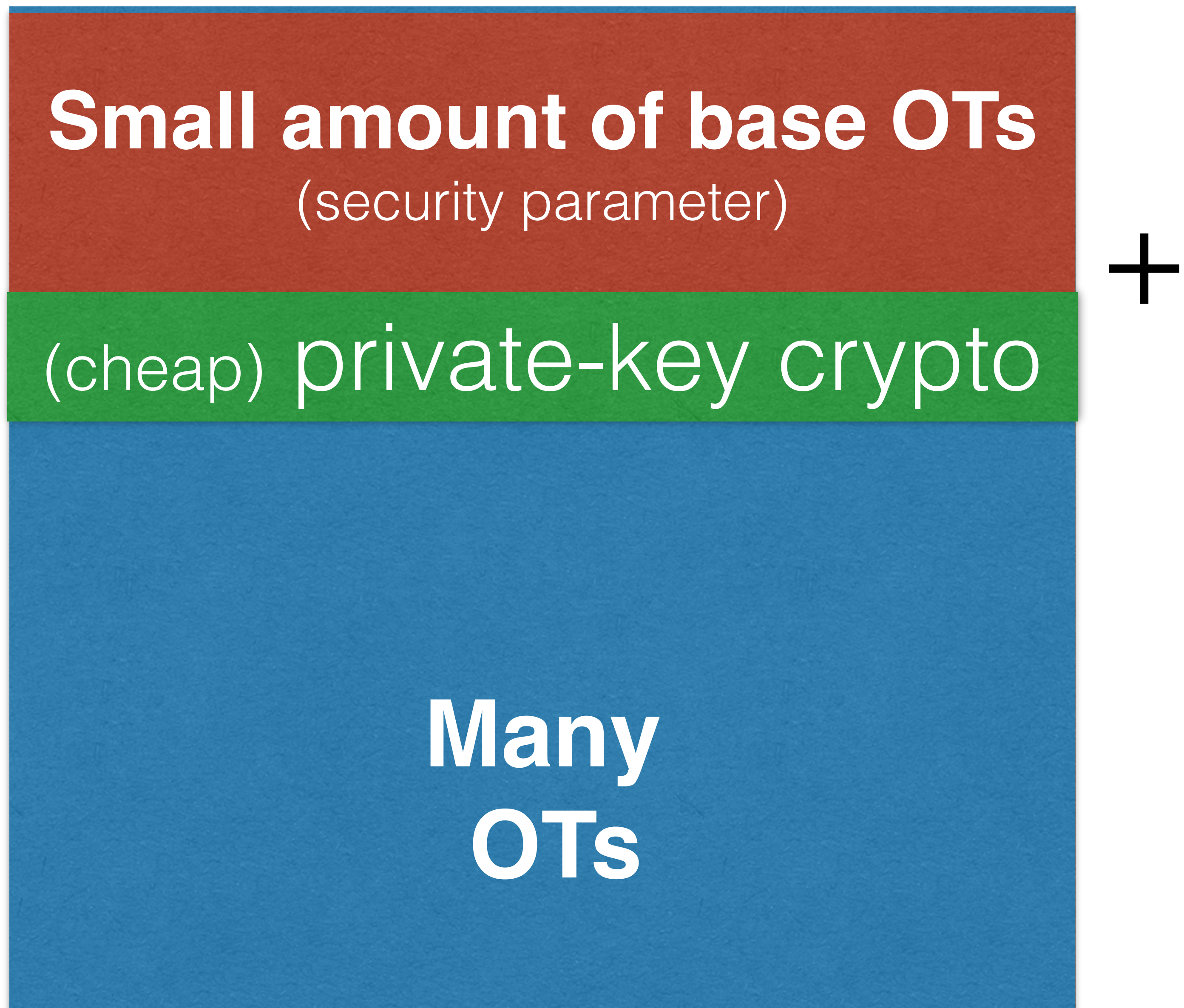
Oblivious Transfer and Secure Computation

- OT is a basic ingredient in (almost) all protocols for secure computation
- **Protocols based on Garbled Circuits (Yao):**
  1 OT per ***input***
  [LP07,LPS08,PSSW09,KSS12,FN13,SS13,LR14,HKK+14,FJN14]
- **Protocols based on GMW:**
  1+ OT per ***AND-gate***
  TinyOT [NNOB12,LOS14] MiniMac protocols [DZ13,DLT14]

How Many OT's?

- **The AES circuit:** Uses 2^{19} OTs
  (when evaluated with TinyOT)
- **The PSI circuit:** (for $b=32, n=2^{16}$) Uses 2^{30} OTs
  (when evaluated with TinyOT)
- Using [P_{eikert}V_{aikuntanathan}W_{aters}08]: 350 OTs per second
  - 1M (2^{20}) OTs > 45 minutes(!)
  - 1G (2^{30}) OTs > 45000 minutes > 1 month…
- [C_{hou}O_{rlandi}15] - 10000 OTs per second (?)

OT Extensions



OT Extension and Related Work

- Introduced in [Beaver96]
- Ishai, Kilian, Nissim, Petrank [IKNP03]
  “Extending Oblivious Transfer Efficiently”
- Optimizations semi-honest: [KK13, ALSZ13]
- Optimizations malicious:
  [Lar14,NNOB12,HIKN08,Nie07]

This Work

- Efficient protocol for OT extension, malicious adversary, based on IKNP
- It outperforms all previous constructions
- Optimizations, implementation

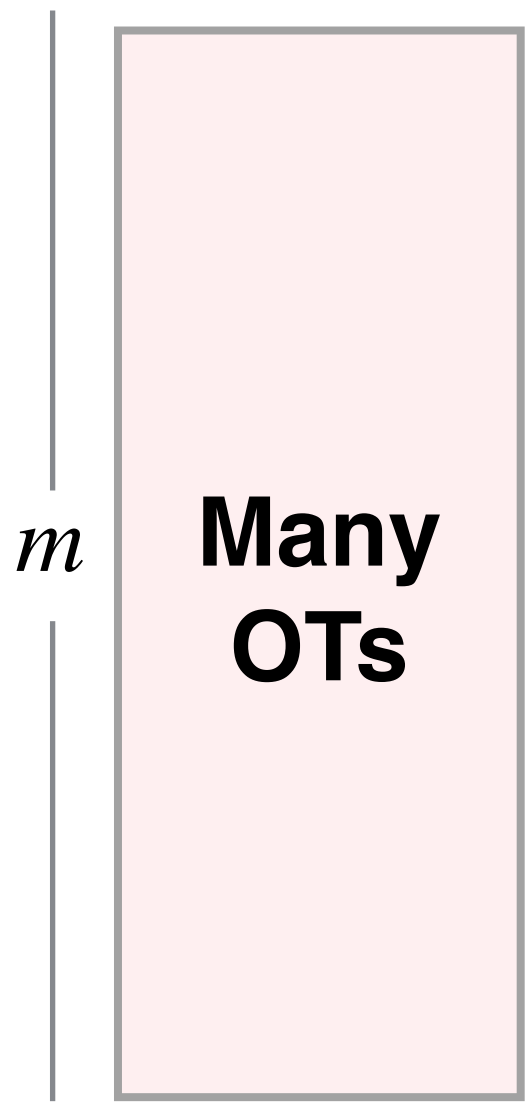
- This Talk:
  - IKNP protocol
  - Our protocol, its security
  - (Implementation) and performance

Extending OT Efficiently[1]

[IKNP03]

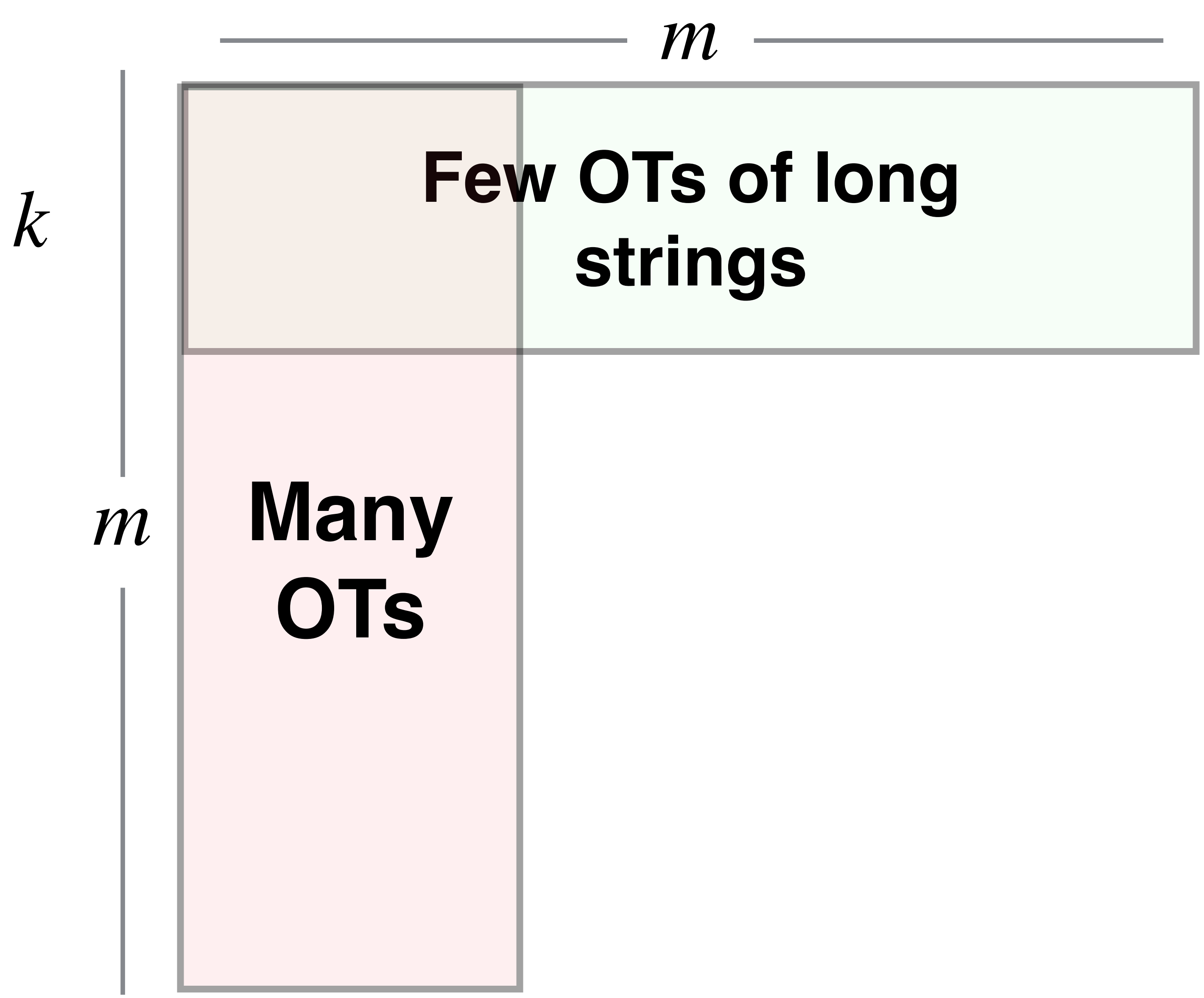
[1]Semi-honest

IKNP - Idea

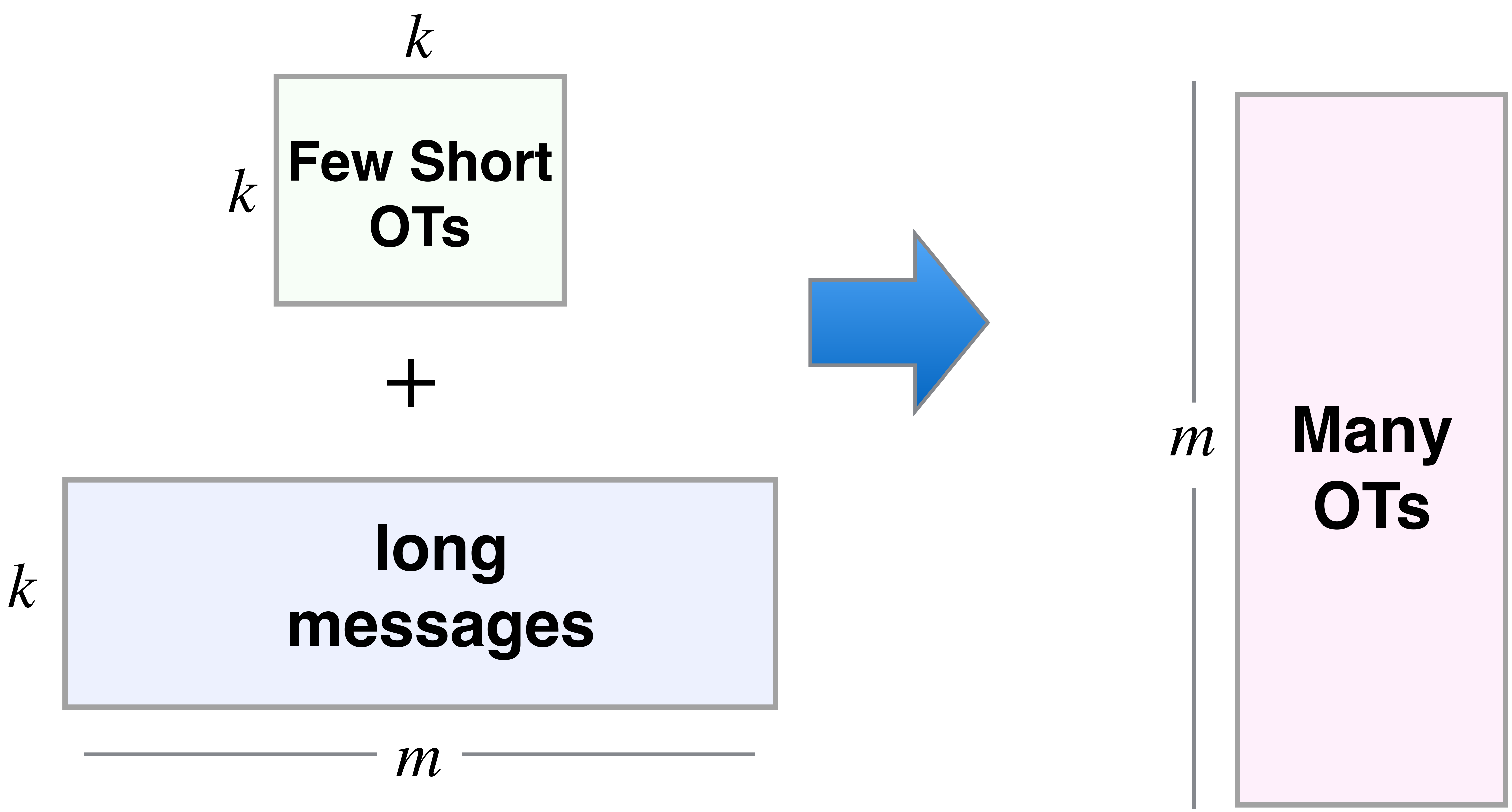


expensive

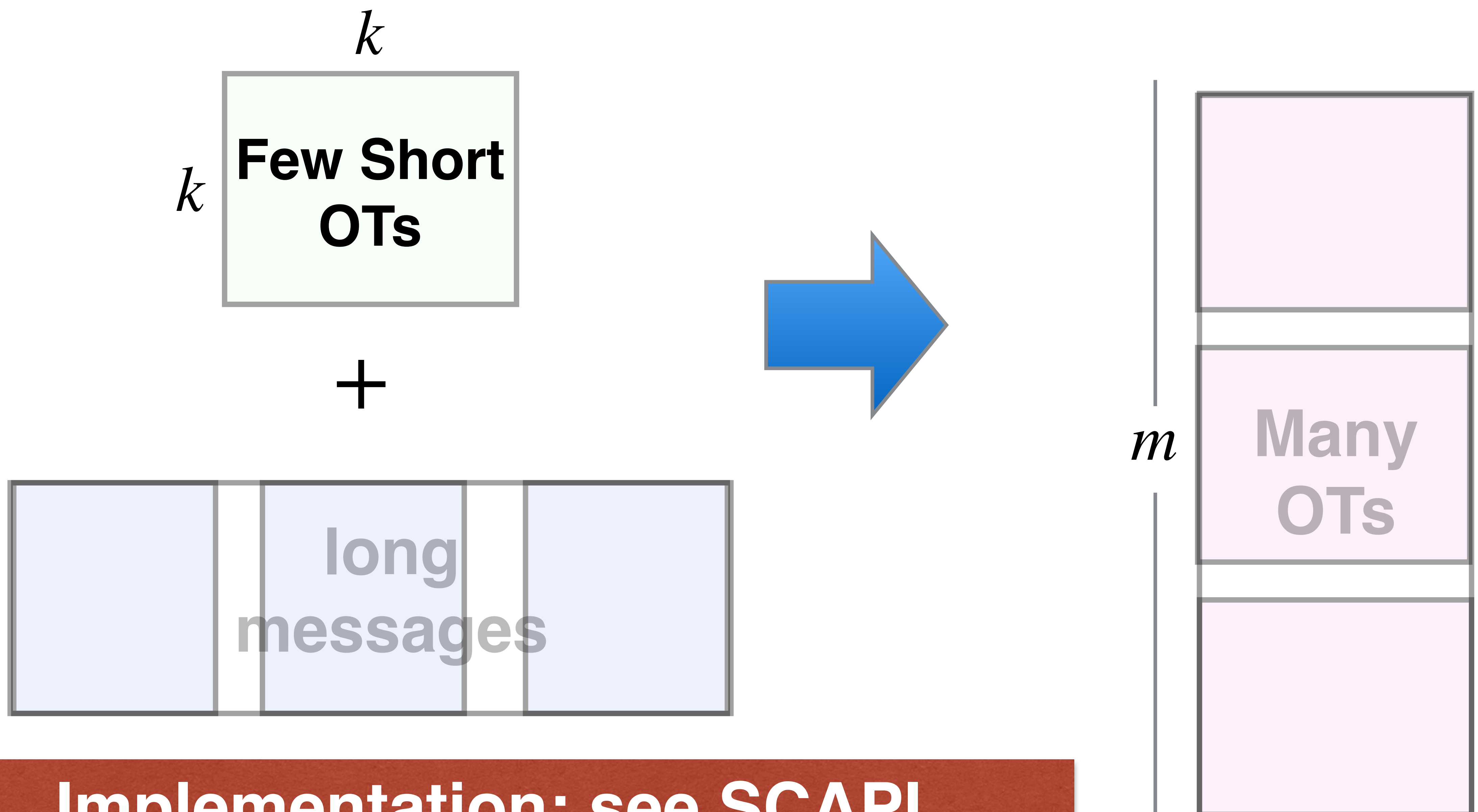
IKNP - Idea



IKNP - Implementation



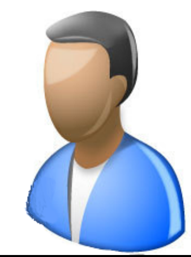
In Practice [ALSZ,CCS13]



Implementation: see SCAPI
https://github.com/cryptobiu/scapi

IKNP

$\{x_j^0, x_j^1\}_{j=1}^{m}$ 

 $\mathbf{r} = (r_1,\ldots,r_m)$

$\mathbf{s} = (s_1,\ldots,s_\ell)$

$\mathbf{k}_1^{s_1},\ldots,\mathbf{k}_\ell^{s_\ell}$

Base OTs

$\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$

$\xleftarrow{\mathbf{u}^1,\ldots,\mathbf{u}^\ell}$

$\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$

$\mathbf{Q}$

$\mathbf{T}$

*

*

$y_j^0 = x_j^0 \oplus H(\mathbf{q}_j)$

$y_j^1 = x_j^1 \oplus H(\mathbf{q}_j \oplus \mathbf{s})$

$\xrightarrow{y_j^0, y_j^1}$

When Moving to Malicious

- The protocol is already secure with respect to malicious **Sender**
- The **Receiver** sends many messages of the same form

$$\xleftarrow{\mathbf{u}^1,\ldots,\mathbf{u}^\ell} \qquad \mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$$

- Security against malicious **Receiver**: we must guarantee that it uses the **same** value **r** in these messages

The Protocol

$\{x_j^0, x_j^1\}_{j=1}^{m}$

$\mathbf{r} = (r_1, \ldots, r_m)$

Base OTs

$\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$

$\xleftarrow{\mathbf{u}^1, \ldots, \mathbf{u}^\ell}$

$\mathbf{Q}$ $\mathbf{T}$

Consistency Check of $\mathbf{r}$

$y_j^0 = x_j^0 \oplus H(\mathbf{q}_j)$

$y_j^1 = x_j^1 \oplus H(\mathbf{q}_j \oplus \mathbf{s})$

$\xrightarrow{y_j^0, y_j^1}$

The Consistency Checks

Consistency Check

$$\mathbf{u}^{i} = G(\mathbf{k}_{i}^{0}) \oplus G(\mathbf{k}_{i}^{1}) \oplus \mathbf{r}$$

$$\mathbf{u}^{j} = G(\mathbf{k}_{j}^{0}) \oplus G(\mathbf{k}_{j}^{1}) \oplus \mathbf{r}$$

Consistency Check

$$\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$$
$$\mathbf{u}^j = G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}$$

$$\oplus \begin{array}{l} \mathbf{u}^i = \mathbf{t}_i^0 \oplus \mathbf{t}_i^1 \oplus \mathbf{r} \\ \mathbf{u}^j = \mathbf{t}_j^0 \oplus \mathbf{t}_j^1 \oplus \mathbf{r} \end{array}$$

$$\mathbf{u}^i \oplus \mathbf{u}^j = \mathbf{t}_i^0 \oplus \mathbf{t}_i^1 \oplus \mathbf{t}_j^0 \oplus \mathbf{t}_j^1$$

$$\boxed{\mathbf{u}^i \oplus \mathbf{u}^j \oplus \mathbf{t}_i^{s_i} \oplus \mathbf{t}_j^{s_j}} \; ?= \; \boxed{\mathbf{t}_i^{1-s_i} \oplus \mathbf{t}_j^{1-s_j}}$$

$$\mathrm{H}(\mathbf{u}^i \oplus \mathbf{u}^j \oplus \mathbf{t}_i^{s_i} \oplus \mathbf{t}_j^{s_j}) \; ?= \; \mathrm{H}(\mathbf{t}_i^{1-s_i} \oplus \mathbf{t}_j^{1-s_j})$$

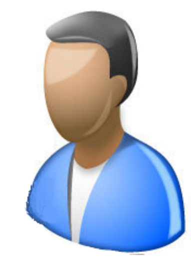
Consistency Check

$$h_{i,j}^{0,0} = H(\mathbf{t}_i^0 \oplus \mathbf{t}_j^0)$$

$$h_{i,j}^{0,1} = H(\mathbf{t}_i^0 \oplus \mathbf{t}_j^1)$$

$$h_{i,j}^{1,0} = H(\mathbf{t}_i^1 \oplus \mathbf{t}_j^0)$$

$$h_{i,j}^{1,1} = H(\mathbf{t}_i^1 \oplus \mathbf{t}_j^1)$$



For every pair (i,j)

$$\xleftarrow{\hspace{10em}}$$

$$\mathbf{u}^1,\ldots,\mathbf{u}^\ell \quad \{h_{i,j}^{0,0}, h_{i,j}^{0,1}, h_{i,j}^{1,0}, h_{i,j}^{1,1}\}_{i,j}$$

Alice checks that every pair *(i,j)*:

$$h_{i,j}^{1-s_i,1-s_j} \;?\!= H(\mathbf{u}^i \oplus \mathbf{u}^j \oplus \mathbf{t}_i^{s_i} \oplus \mathbf{t}_j^{s_j})$$

$$h_{i,j}^{s_i,s_j} \;?\!= H(\mathbf{t}_i^{s_i} \oplus \mathbf{t}_j^{s_j})$$

Does it work?

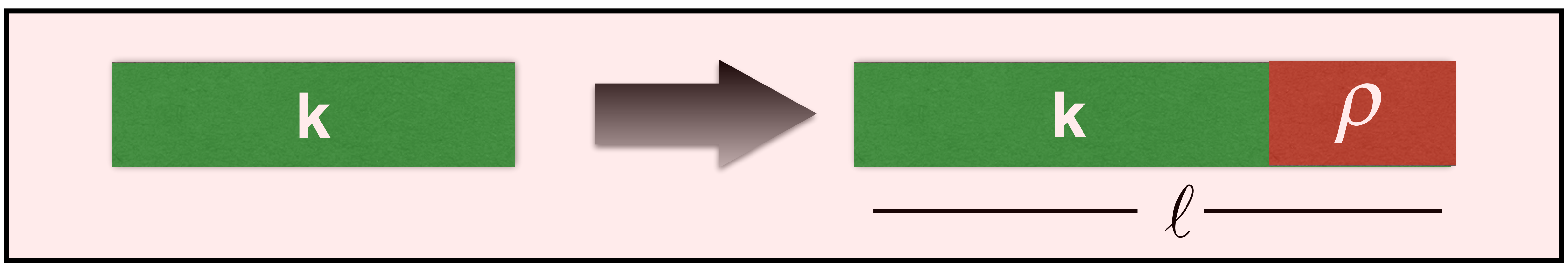
- **Our check is not sound:**
  - The adversary can still send $\mathbf{u}^i$, $\mathbf{u}^j$, with $\mathbf{r}^i \neq \mathbf{r}^j$
  - But, it takes a risk…
  - Effectively, in order to pass the verification of (i,j) it has to guess either $\mathbf{s}_i$ or $\mathbf{s}_j$

- Our check guarantees the following:

If the adversary tries to cheat with u^i, u^j it gets caught with probability 1/2!

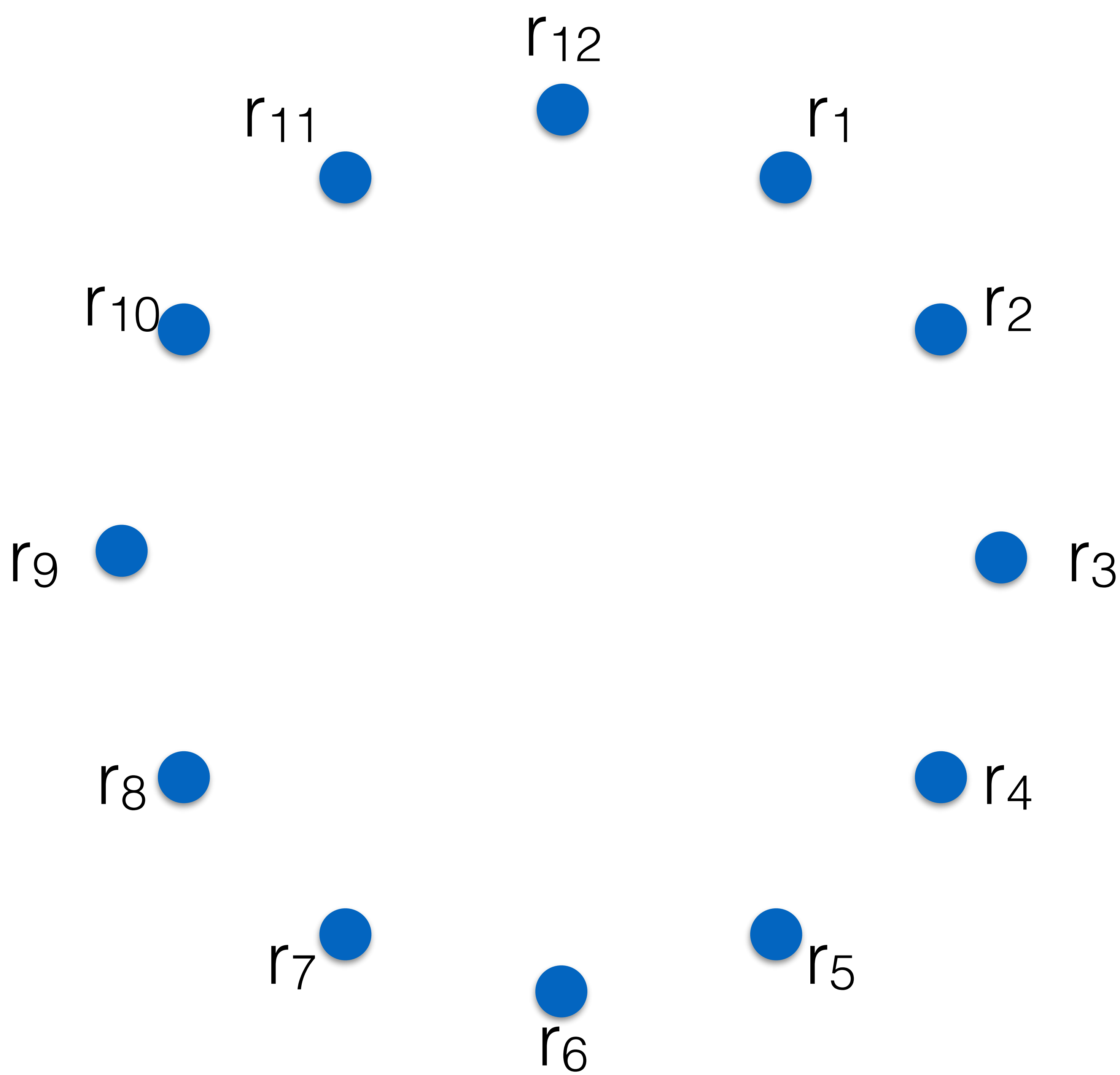
Consistency Check

- **Receiver** cannot cheat in many messages
  - with each cheat - one bit of **s** is leaked
  - **s** is the "secret key" of the sender
- **Solution** - increase the size of **s**

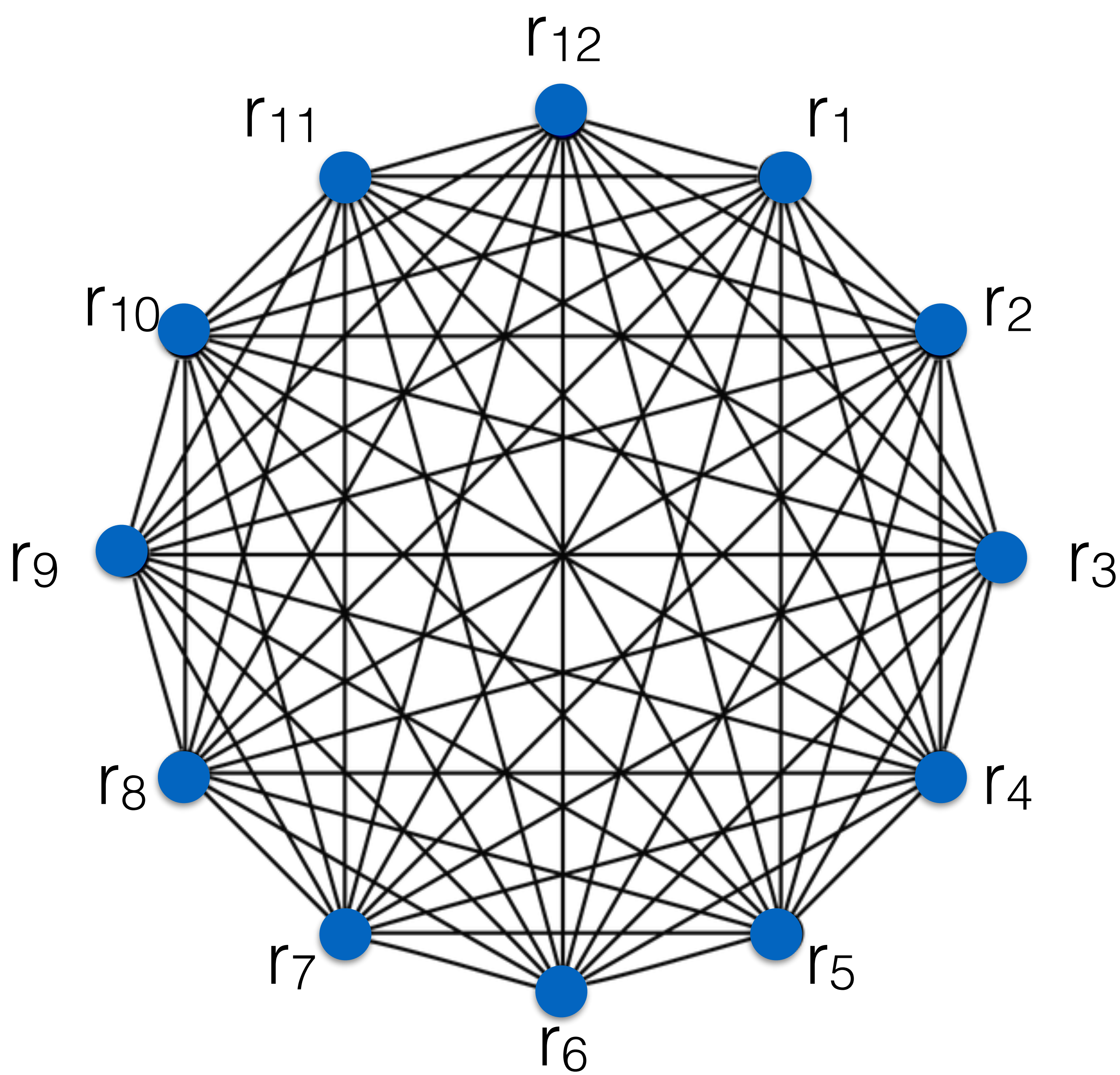


- But wait… you have ℓ^2 amount of checks
  Do we really need this huge amount of checks?

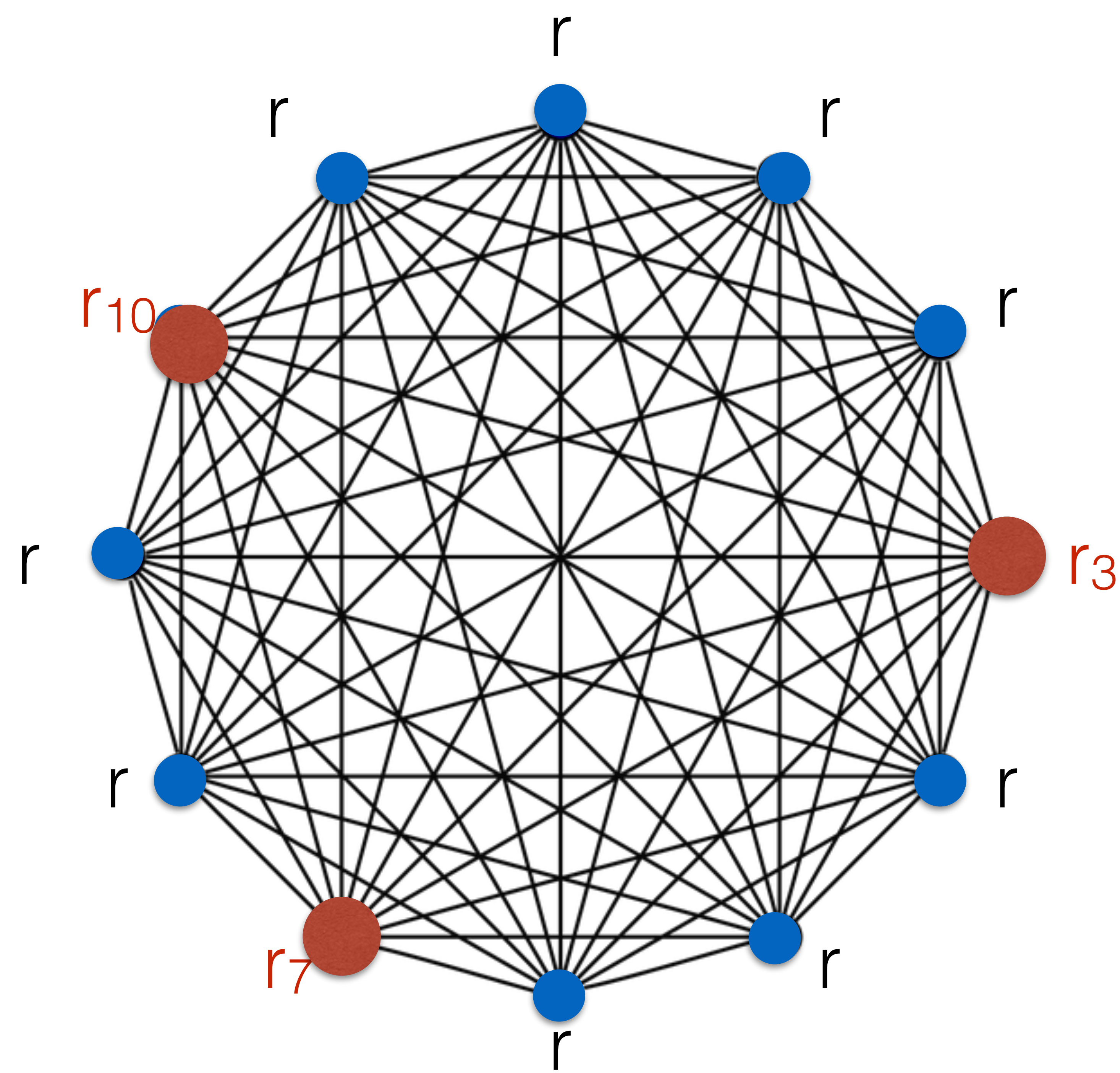
How many checks do we really need?



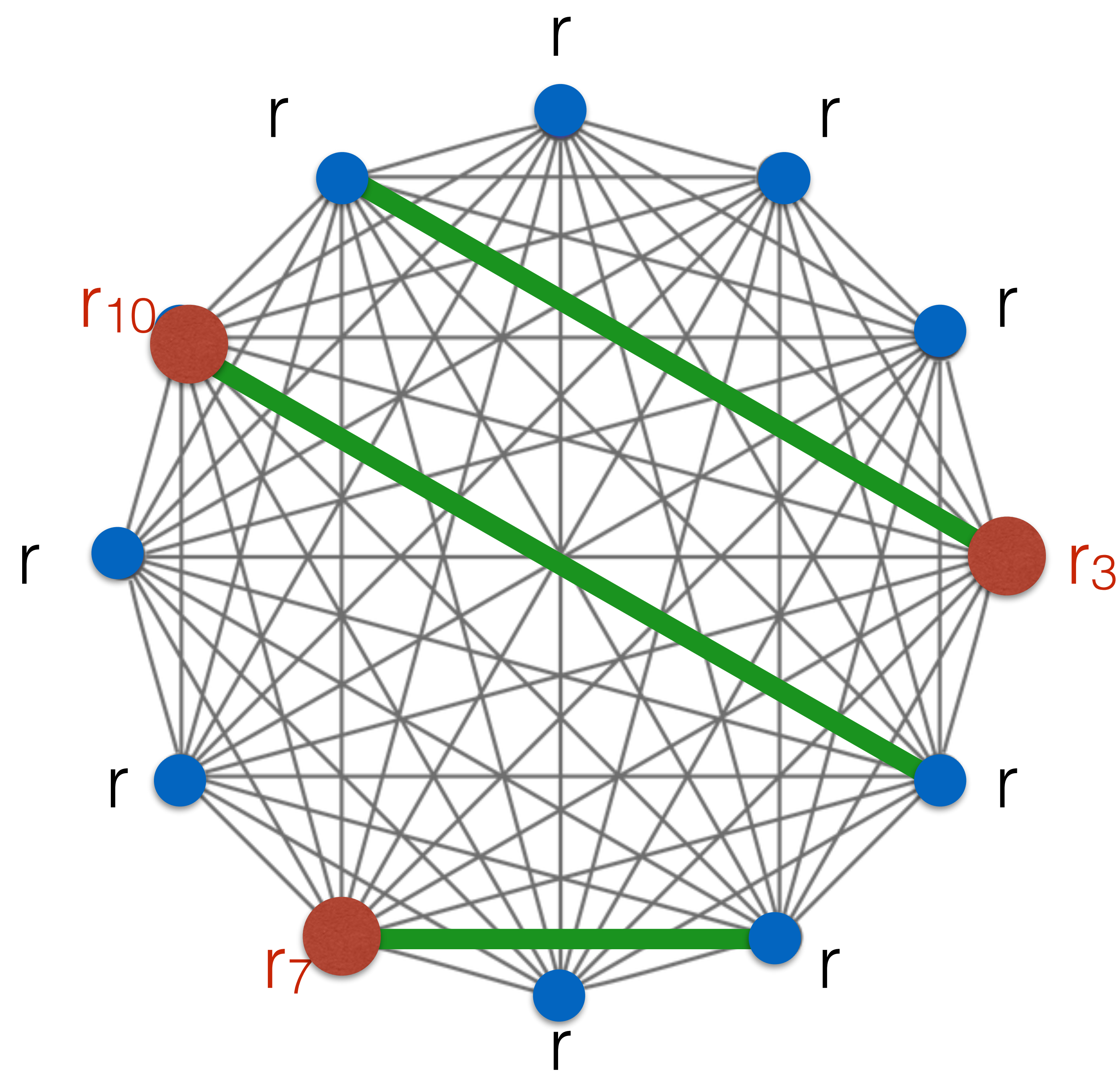
How many checks do we really need?



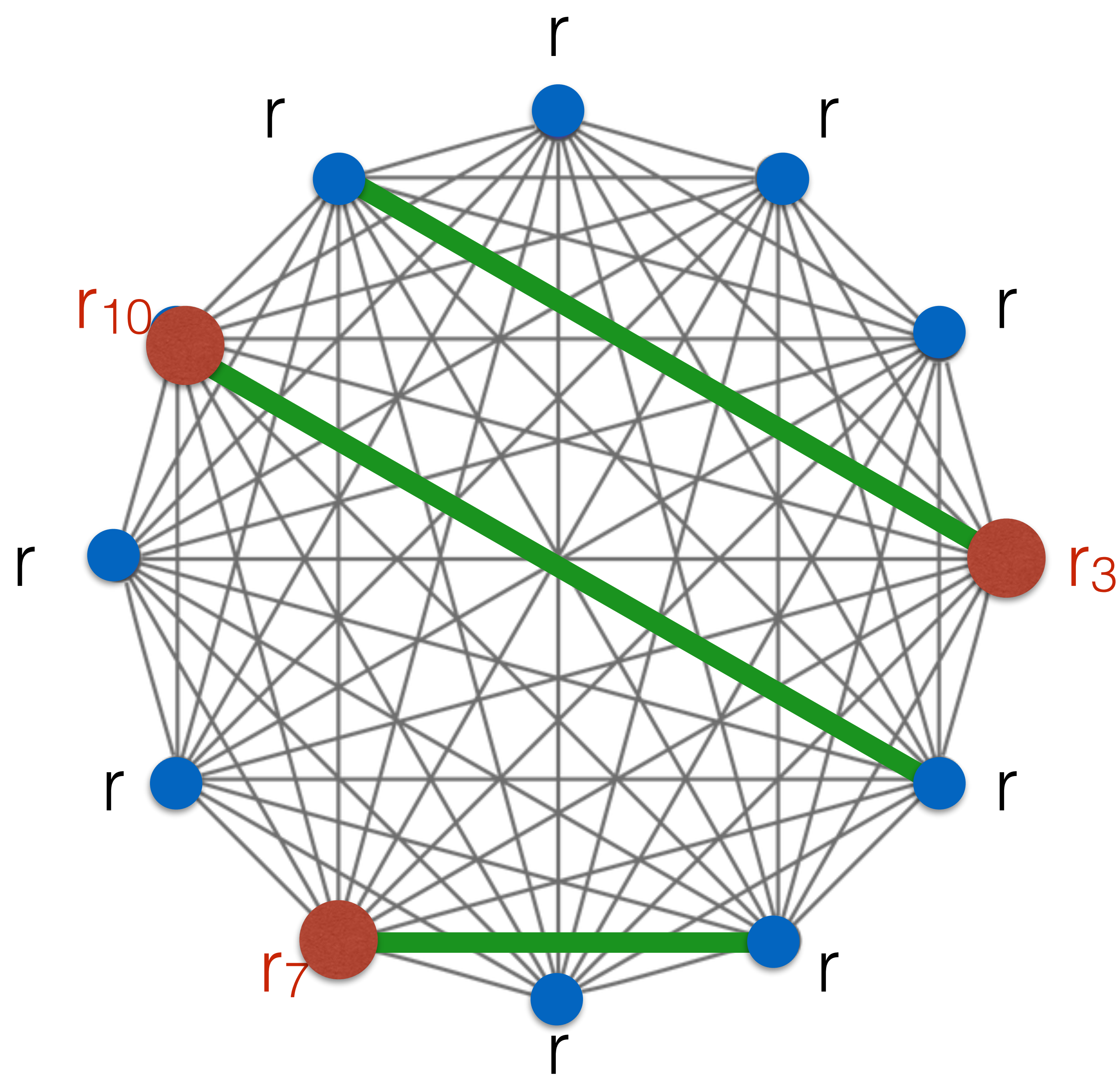
How many checks do we really need?



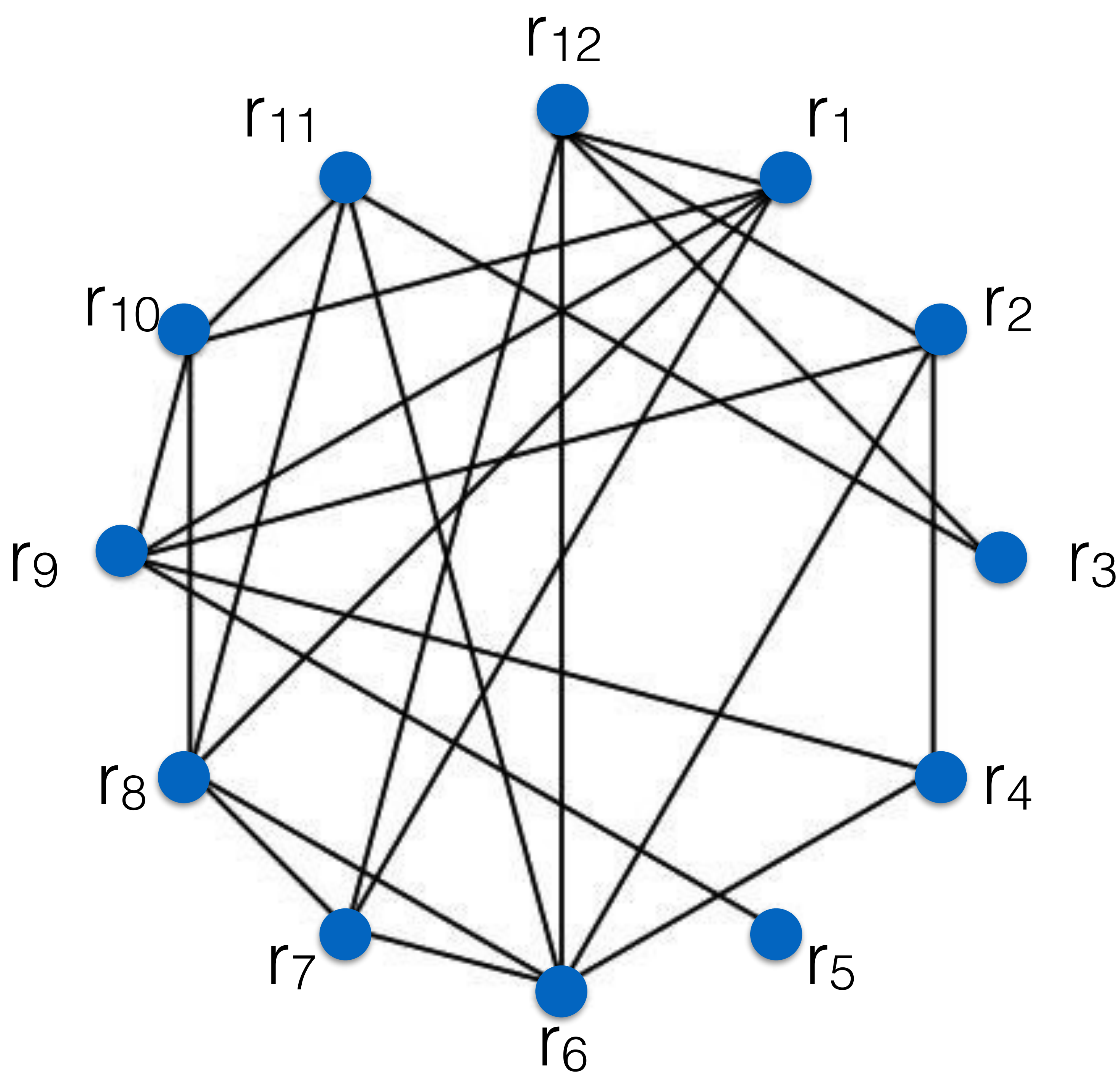
How many checks do we really need?



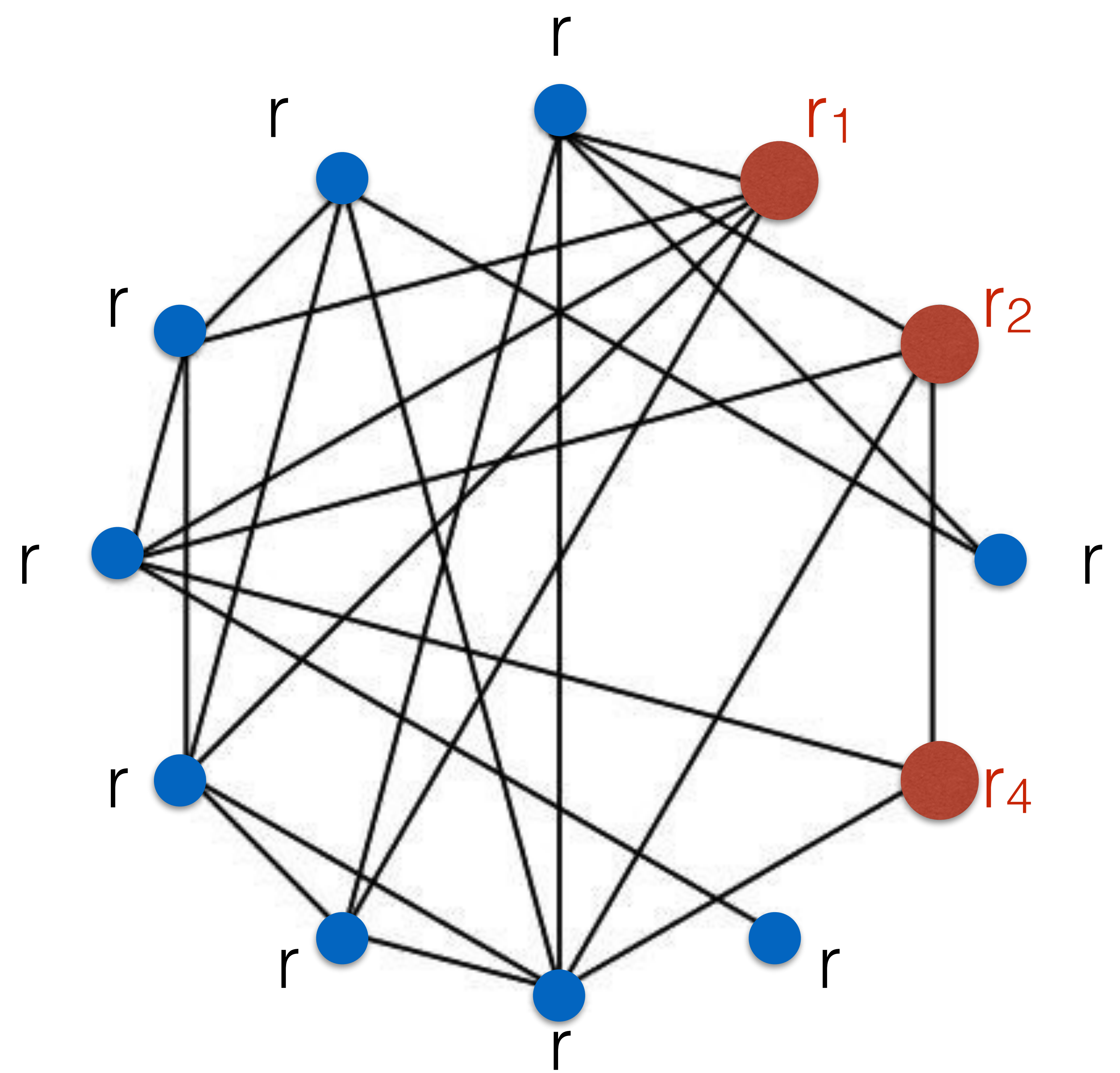
The needed property:
For any “large enough" set of **bad** vertices
(> p=40), **there exists** p-matching with the **good** vertices



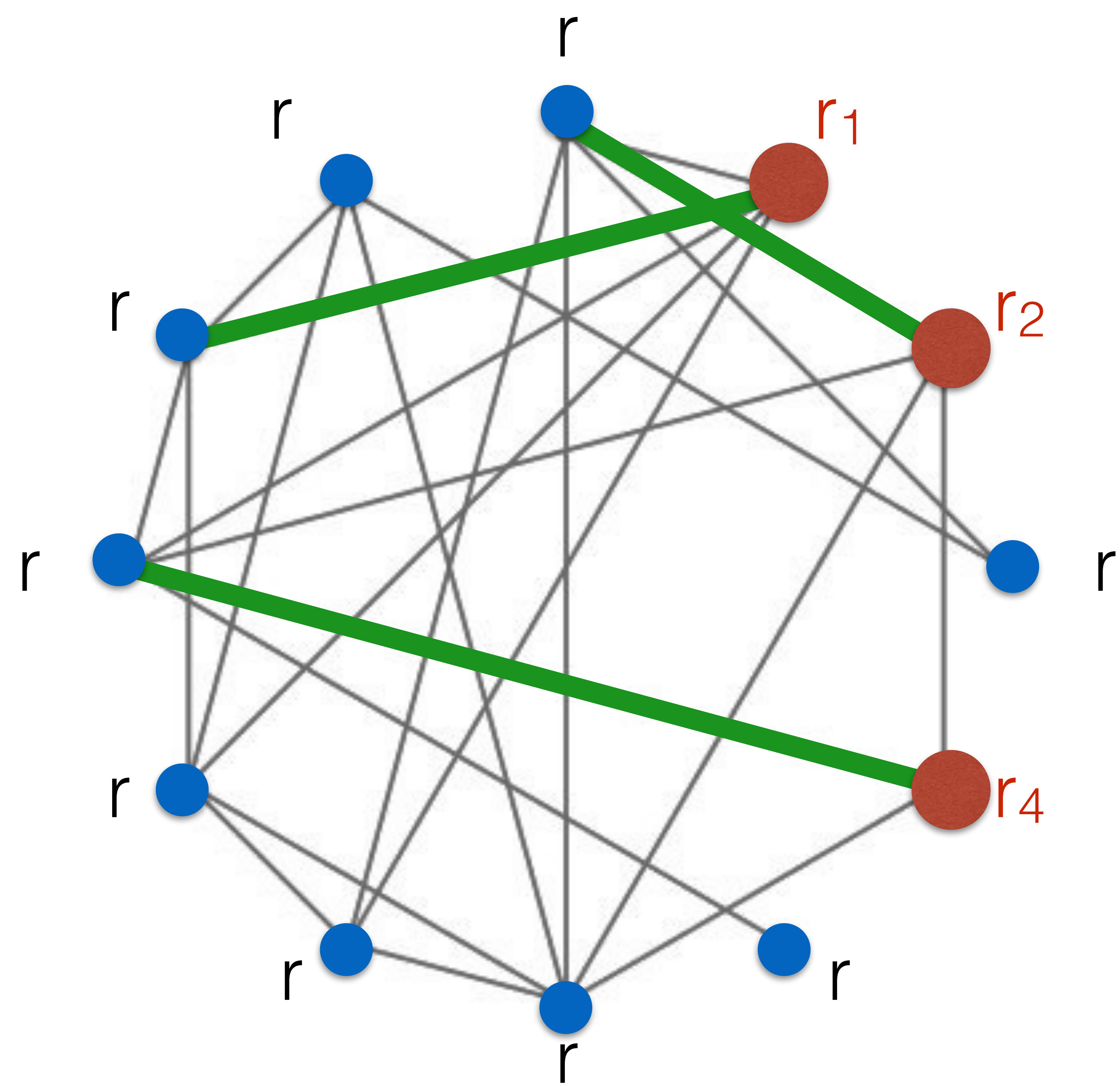
How many checks do we really need?



How many checks do we really need?



How many checks do we really need?



How Many Checks?

The needed property:
For any "large enough" set of **bad** vertices (> *p*=40), **there exists** *p*-matching with the **good** vertices

- We show that random **d-regular graph** satisfies the above (for appropriate set of parameters)
  - For k=128, p=40
    - 168 base OTs, complete graph: 14028
    - 190 base OTs, d=2, checks: 380
    - 177 base OTs, d=3, checks: 531

- **Covert**: (168 base OTs) probability 1/2, just random 7 checks!

Instantiation of **H**

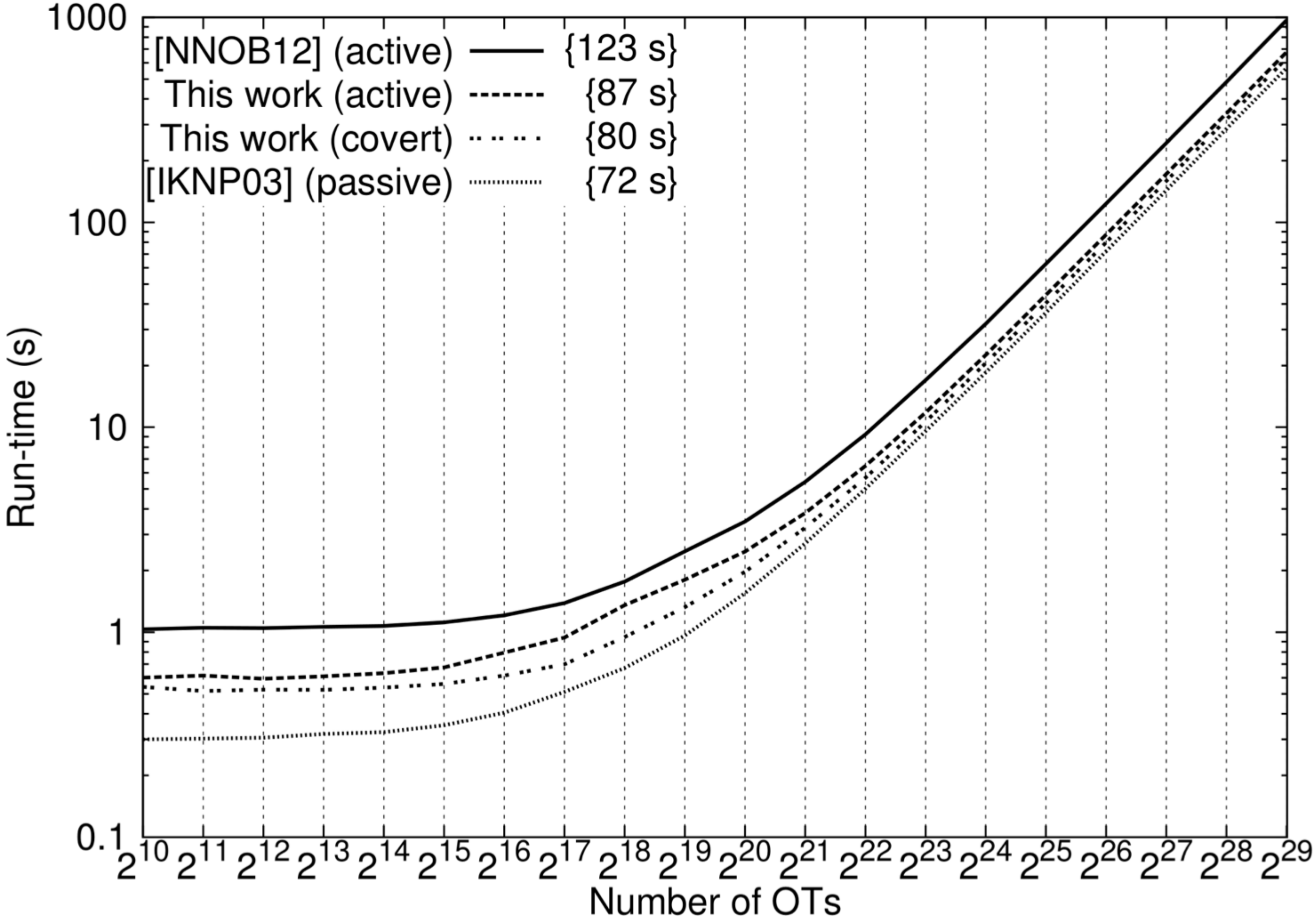
- [IKNP] assumes that **H** is **Correlation-Robust**
- Sometimes, in order to gain more efficiency, protocols need some stronger properties of **H**, and so it is assumed to be a **Random-Oracle**
- **Correlation-robustness** is much more plausible assumption than **random-oracle**
- We have some leakage of **s**, and so **H** is assumed to be **Min-Entropy Correlation Robustness**

Performance

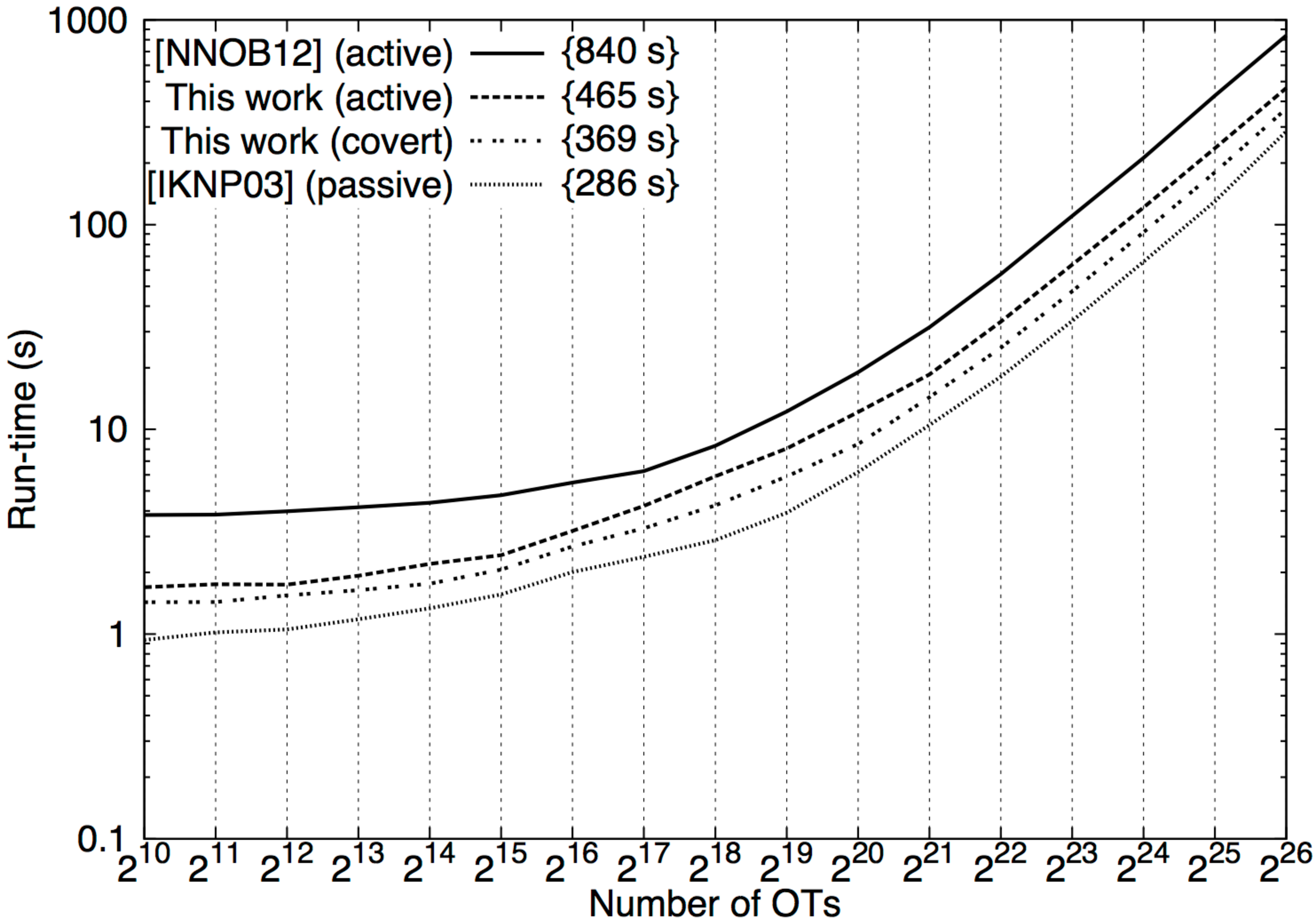
Empirical Evaluation

- Benchmark: 2^{23}=8M OTs
- **Local scenario (LAN):**
  Two servers in the same room
  (network with low latency and high bandwidth)
  12 sec (190 base OTs, 380 checks)
- **Cloud scenario (WAN):**
  Two servers in different continents
  (network with high latency and low bandwidth)
  64 sec (174 base OTs, 696 checks)

Comparison - LAN Setting



Comparison - WAN setting



Conclusions

- More efficient OT extension - more efficient protocols for MPC
- Optimized OT extension protocol, malicious adversary
- Combination of theory and practice

Thank You!