UNBOUND

WHERE SECURITY IS KEY

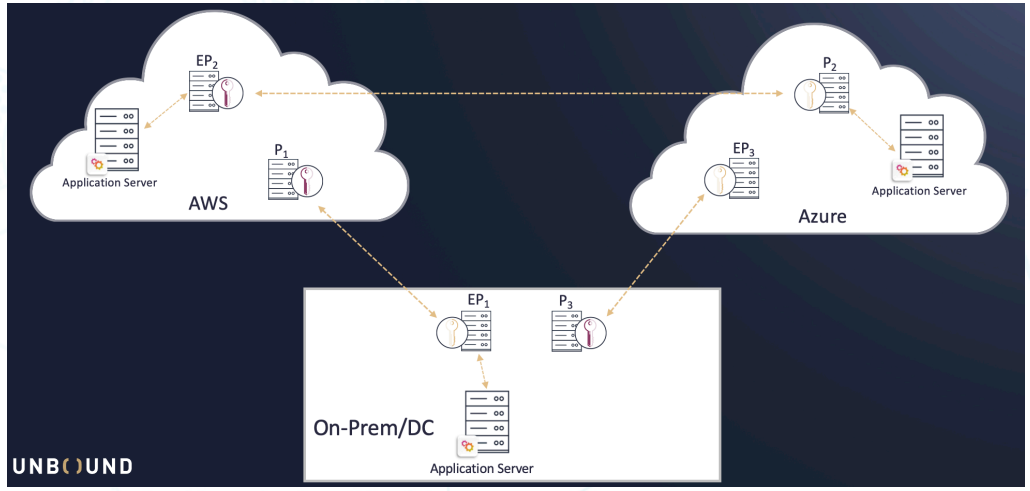
Lessons and Challenges in Deploying (Heavy) MPC in Different Environments

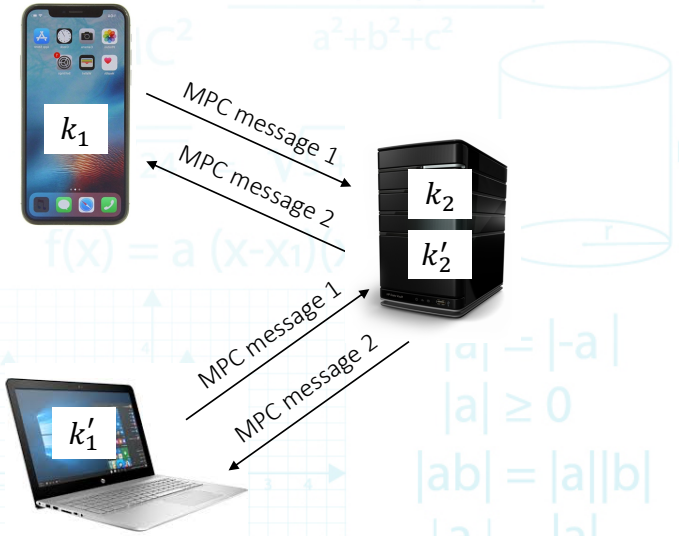
Yehuda Lindell, Valery Osheter and Samuel Ranellucci

Unbound Tech

Background and Context

- **Unbound provides key management and key protection in software**
  - Keys are protected via MPC; shared between two (or more) machines
- **Product lines (of relevance)**
  - Server-side key protection – functions as a virtual HSM
  - Endpoint key protection – functions as a virtual smartcard or token





Service vs Enterprise Software

- **Many solutions today are offered as SaaS (Software as a Service)**
  - Customer has an API to access the service and doesn't install anything (or at most a small client)
- **Important distinction:**
  - A SaaS company runs its own software
    - You know what machines, what network, what environment – you control all!
  - Enterprise software solutions are run by customers
    - You don't know if it's in the cloud, in a data center, what type of machine, etc.
    - Mandating specific environments reduces your relevance and can increase cost
      - E.g., powerful machines in AWS are **much more expensive** than weak ones

RSA Key Generation

- **Consider deployment of CRYPTO 2018 paper**
  - Fast Distributed RSA Key Generation for Semi-Honest and Malicious Adversaries, by T. Frederiksen, Y. Lindell, V. Osheter and B. Pinkas
- **Impressive performance – average time of 35 seconds for malicious RSA-2048 key generation**
  - Used Intel Xeon E5-2673 v.4 - 2.3Ghz machines with 64Gb RAM, with 40.0 Gbps network
    - These machines have 20 cores and 40 threads
    - The time of 35 seconds is for an 8-thread implementation

Issue 1 – Cost

- **RSA key generation paper used Intel Xeon E5-2673 v.4 - 2.3Ghz machines with 64Gb RAM, with 40.0 Gbps network**
- **These types of machines cost thousands of dollars a year, versus a few hundred dollars for a 2-core, 8GB RAM machine (for example)**
  - When deploying 4 machines in each region over 3 regions, this becomes expensive
- **This significantly influences the product TCO (total cost of ownership)**

Issue 2 – Multithreading

- **RSA key generation naturally tends to multi-threaded computation**
  - Multiple candidates are tested – can be done in parallel
- **Proposal: automatically detect number of cores and parallelize to that number**
  - Or just use many threads and pay overhead when you have less
- **Problem:**
  - The machine is used for many cryptographic operations
  - Using many cores can choke the machine and slow down ongoing operations
    - Database access becomes slow since someone else is generating an RSA key!

Issue 3 – High Bandwidth

- **RSA key generation is very heavy, and requires many iterations to find a modulus**
- **There is a single honesty check at the end**
  - Since most candidates are thrown out, defer a proof of honest to the end
  - This is OK since if someone cheats, it's OK that they learned part of the key
    - The key will be thrown out
- **The proof is run once, and is not significant**
  - Use dual execution on about 5-6 million gates => each party sends and receives about 170MB
  - On a 100Gbps network, would take about 15 seconds

Bandwidth – Virtual Smartcard Setting

- **Generate an RSA key**
  - Between a mobile and server
  - Between a laptop and server
- **The laptop is at someone's home (WFH)**
  - Typical uploads from home are between 1Mbps and 15Mbps
  - At 1 Mbps, uploading 170MB would take 1360 seconds = 23 minutes
    - But often the actual upload can be slower (depending on other usage)
  - This is before we count **anything else**
- **It is absolutely essential to use a shorter proof, like Ligero and variants**
- **Other significant bandwidth savings were made (Paillier instead of OT)**

Additional Changes

- **Numerous local optimizations – won't go through them here**
  - Verify more candidates in parallel in order to reduce variance
  - Better balance of initial filtering to counter speed and communication
  - Implemented fixed-base variant of Paillier and faster exponentiation
  - Boneh-Franklin biprimality test – part 2 – isn't needed
  - Optimizations in circuit for honest check
    - E.g., compare that $N = p \cdot q$ is expensive (needs multiplication and comparison)
      - It is much cheaper to check $N = p \cdot q \bmod r$, since only multiply $p \bmod r$ and $q \bmod r$
    - We need to prove consistency with AES operations (for PRG and commitments); utilize algebraic representation of AES
  - And much more

Engineering

- **Cryptographers shouldn't think that all solutions are cryptographic**
  - Our implementation takes about 5 minutes from home to server
    - This is much better than 20-40 minutes, but could be viewed as still too long
- **Engineering solutions work**
  - RSA key generation is run upon smartcard installation
    - Run in background (like antivirus installation)
    - Much faster than delivering a physical smartcard
  - In server setting, generate many RSA keys and have them in reserve
- **Complicated engineering solutions are less desired**
  - Auto-detect bandwidth and computing power is a problem – you don't know what else may run at the same time, or if there is high variance

Conclusions

- **In non-SaaS settings, it's hard to assume anything**
  - There are big cost advantages to running on small and weak machines
  - Bandwidth can be very fast and very slow
- **Stability and predictability are often more important than speed**
  - High variance is a challenge
- **Not all solutions are cryptographic – a lot is smart engineering design**
- **Academic papers should continue to push the science forward**
  - But actual solutions in production need to find different balances
  - It's important to look at "balanced solutions" that don't necessarily minimize time or bandwidth, etc.


UNBOUND
WHERE SECURITY IS KEY