Searchable Encryption

From Theory to Implementation

Raphael Bost
Direction Générale de l'Armement - Maitrise de l'Information & Université de Rennes 1

ECRYPT NET Workshop - Crypto for the Cloud & Implementation - 28/06/2017

Security vs. Efficiency

If you had one thing to keep from this presentation:

Searchable encryption is all about a security-performance tradeoff

No free lunch …

This presentation

What are the theoretical and practical challenges/open problems in searchable encryption?

- Lower bounds
- Constructions
- Implementation

We will focus on single keyword SE

Security vs. Efficiency

Efficiency:

- Computational complexity
- Communication complexity
- Number of interactions

Security:

- ???

Evaluating the security

- Use the leakage function from the security definitions
  - ✓ Provable security
  - ✗ Very hard to understand the extend of the leakage
- Rely on cryptanalysis: leakage-abuse attacks
  - ✗ Maybe not the best adversary
  - ✓ 'Real world' implications

Evaluating the security

- We just saw (cf. Kenny's talk) attacks on legacy-compatible searchable encryption
- State-of-the-art schemes leak the number of results of a query
  - ➡ Enough to recover the queries when the adversary knows the database [CGPR'15]
  - ➡ Counter-measure: padding (it has a cost)

Index-Based SE [CGKO'06]

- Structured encryption of the reversed index: search queries allow partial decryption
- Search leakage :
  - repetition of queries (search pattern)
  - number of results

Simple Index-Based SE

- Keyword w matches $DB(w) = (ind_1, \dots, ind_n)$.

  $K_w \leftarrow F(K,w)$

  $\forall 1 \leq i \leq n$, $t_i \leftarrow F(K_w,i)$, $EDB[t_i] \leftarrow Enc(K_w,ind_i)$

- *Search(w)*: the client sends $F(K,w)$ to the server

Efficiency of the scheme

$\forall 1 \leq i \leq |DB(w)|$, $t_i \leftarrow F(K_w, i)$, $EDB[t_i] \leftarrow Enc(K_w, ind_i)$

- Optimal computational and communication complexity
- A lot slower than legacy-compatible constructions !

- t_i's are random ➡ random accesses
  Legacy-compatible ➡ sequential accesses

- Sequential accesses are free after the first one

Locality of SE

- To be competitive with unencrypted databases, SE schemes must have good locality.
- We do not want to access to much data.
  Need of good read efficiency.
- Storage is expensive: low storage overhead is required.

Locality of SE

- Bad news!
  It is impossible to achieve security, constant locality, constant read efficiency and optimal storage all at the same time [CT'14].
- The lower bound is tight [ANSS'16] (good news?).
- Explicit security-performance tradeoff.

Dynamic Index-Based SE

- You might want to update your database. How to add new documents?

  $\forall 1 \leq i \leq |DB(w)|,\ t_i \leftarrow F(K_w, i),\ EDB[t_i] \leftarrow Enc(K_w, ind_i)$

- To insert the entry (w, ind), the client:
  - retrieves $n = |DB(w)|$ (stored on the server)
  - computes $t_{n+1} \leftarrow F(K_w, n+1)$, $c \leftarrow Enc(K_w, ind_i)$
  - sends (t_{n+1}, c)
- Update leakage: repetition of updated keywords

File injection attacks [ZKP'16]

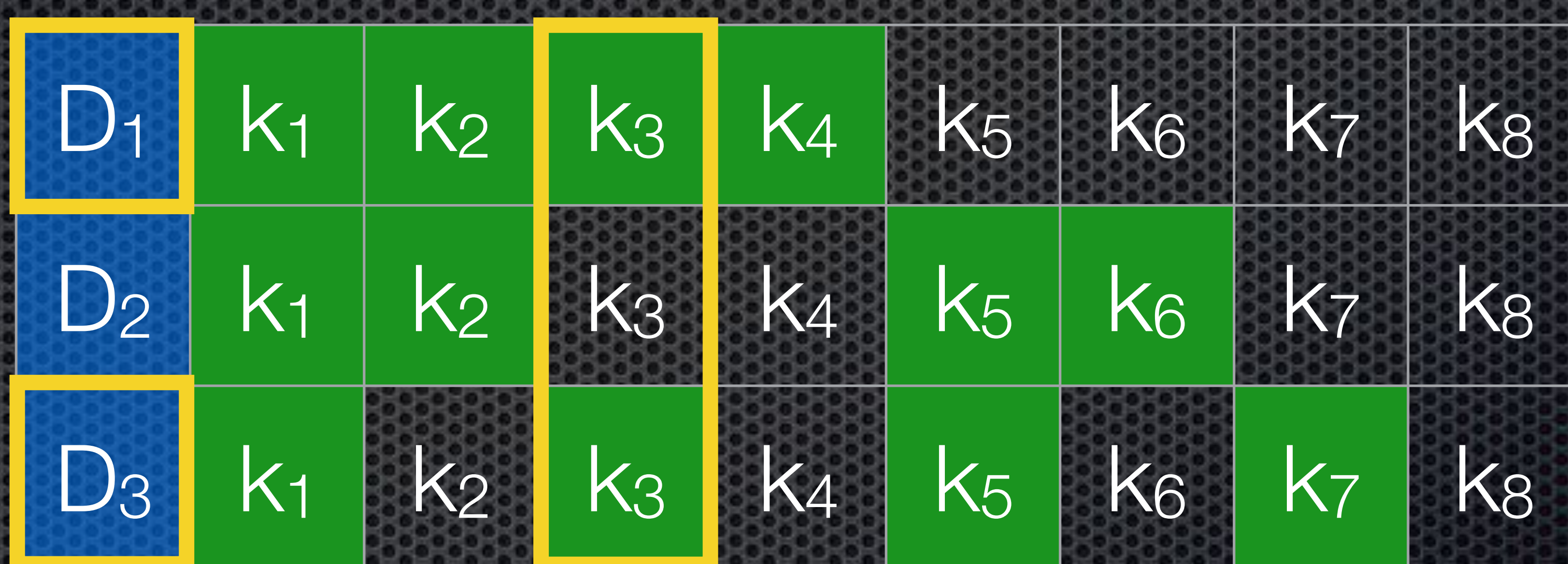
- 'With great power comes great responsibility.'

  Uncle Ben

- New features means new abilities for the attacker.
- The adversary can now be active and insert his own documents (e.g. emails).

File injection attacks [ZKP'16]

- Insert purposely crafted documents in the DB. Use binary search to recover the query



log K injected documents

File injection attacks [ZKP'16]

- Insert purposely crafted documents in the DB.
  Use binary search to recover the query
  - ➡ *$\log K$* injected documents
- Counter-measure: no more than *T* kw./doc.
  - ➡ *$(K/T) \cdot \log T$* injected documents to attack
- Adaptive version of the attack
  - ➡ *$(K/T) + \log T$* injected documents to attack
  - ➡ *$\log T$* injected documents with prior knowledge

‘Active’ Adaptive Attacks

- All these adaptive attacks use the update leakage:
  - For an update, most SE schemes leak if the inserted document matches a previous query
  - We need SE schemes with oblivious updates

Forward Privacy

Forward Privacy

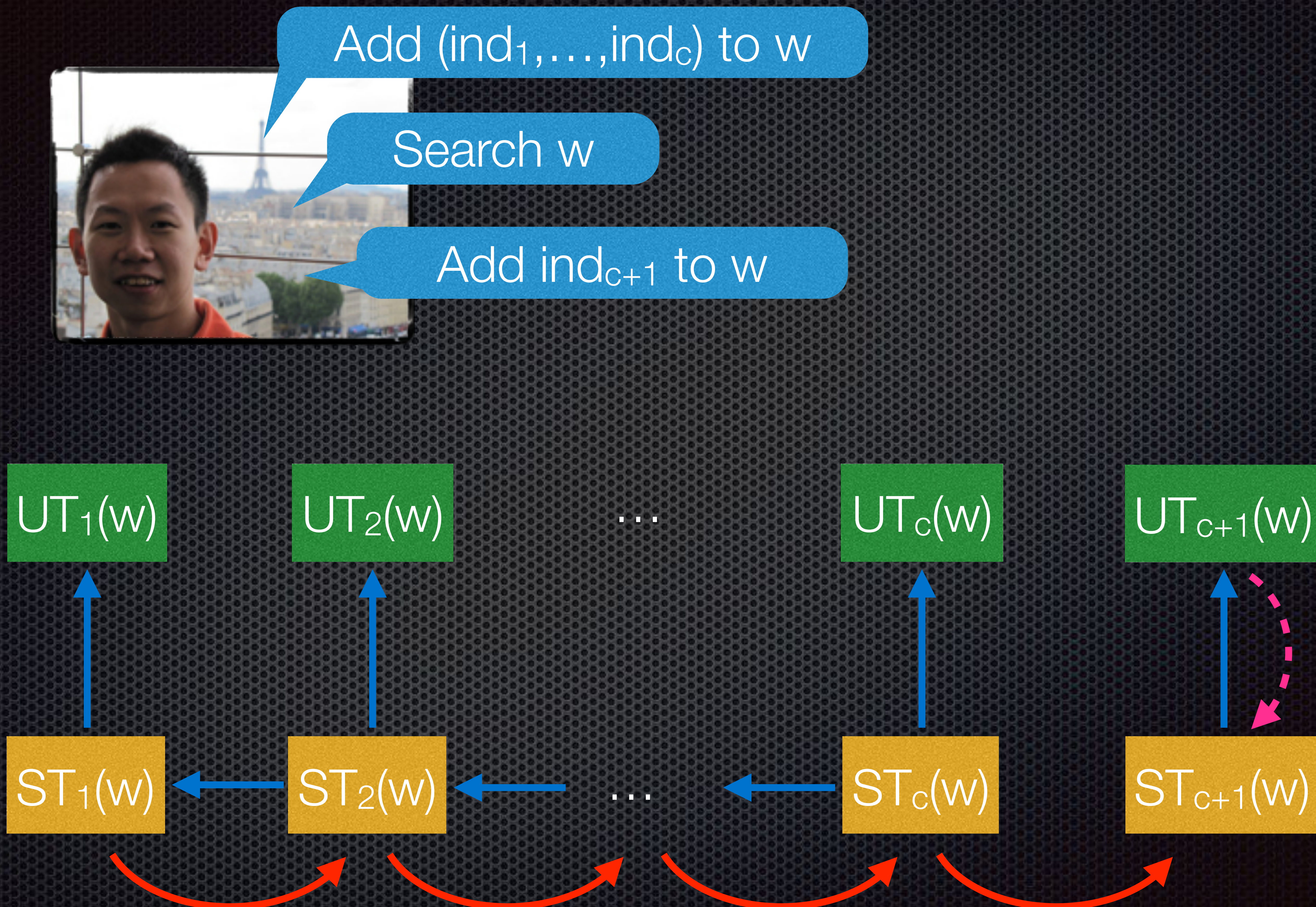
- Forward private: an update does not leak any information on the updated keywords (often, no information at all)
- Secure online build of the EDB
- Only one scheme existed so far [SPS'14]
  - ➡ ORAM-like construction
  - ✗ Inefficient updates: $O(\log^2 N)$ comp., $O(\log N)$ comm.
  - ✗ Large client storage: $O(N^\varepsilon)$

Σοφος

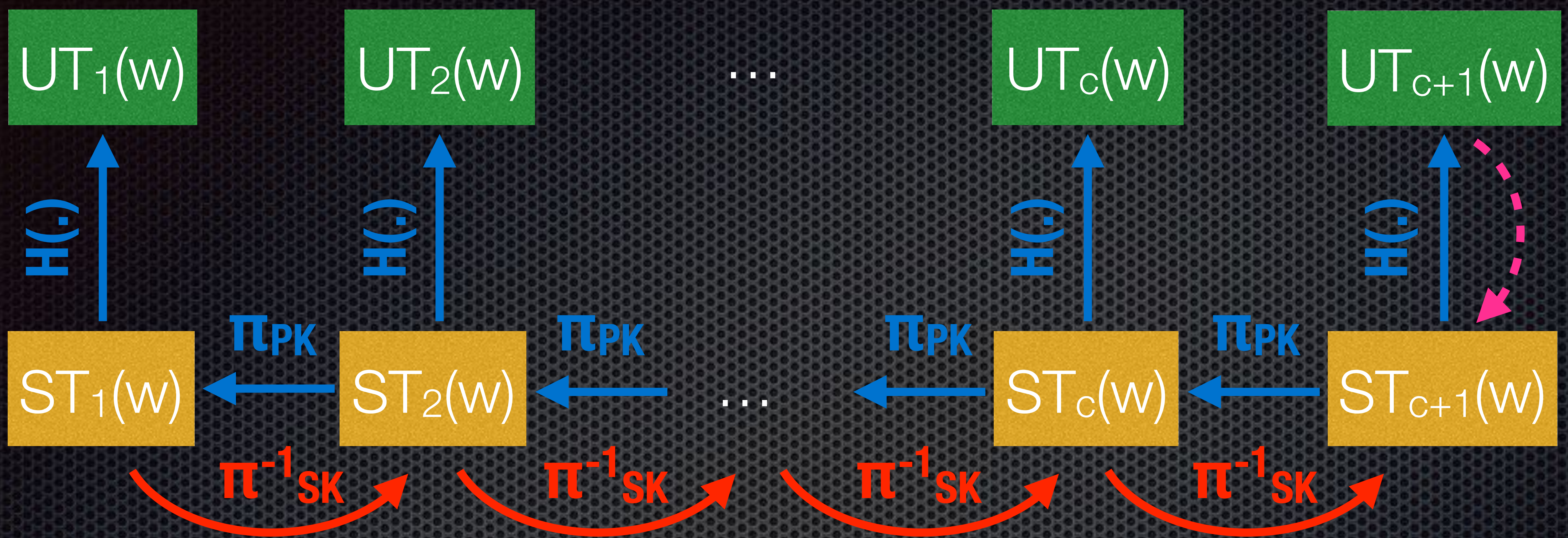
- Forward private index-based scheme
- Low overhead for search and update
- A lot simpler than [SPS'14]

Add (ind$_1$,…,ind$_c$) to w

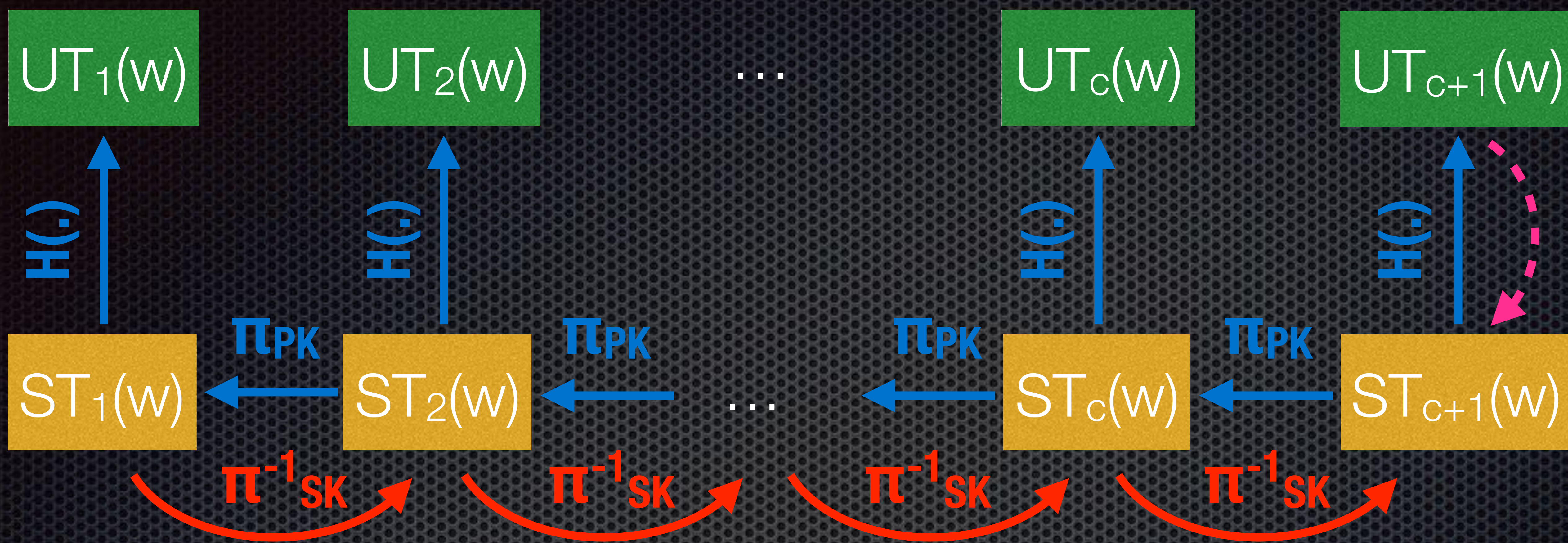
Search w



Add (ind1,…,indc) to w
Search w
Add indc+1 to w
UT1(w)
UT2(w)
…
UTc(w)
UTc+1(w)
ST1(w)
ST2(w)
…
STc(w)
STc+1(w)



- Naïve solution: $ST_i(w) = F(K_w,i)$, send all $ST_i(w)$'s
  - ✗ Client needs to send c tokens
  - ✗ Sending only K_w is <u>not</u> forward private
- Use a trapdoor permutation



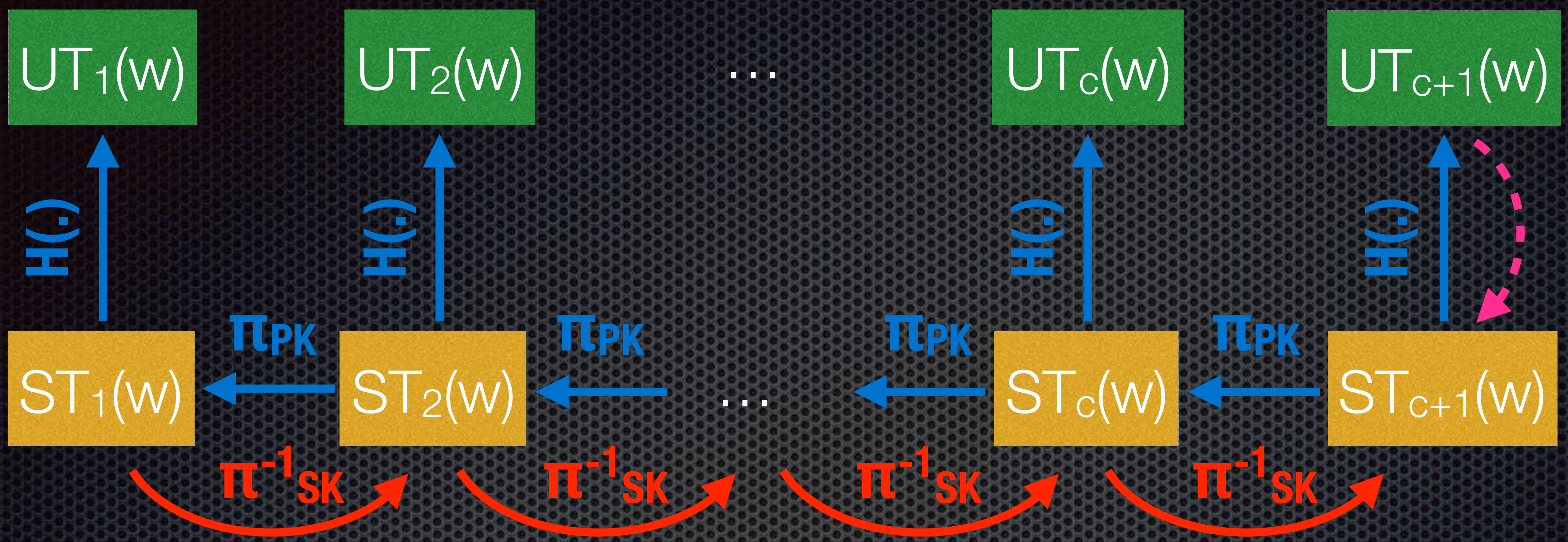
Search:

- Client: *O(1)*
- Server: *O(|DB(w)|)*

Update:

- Client: *O(1)*
- Server: *O(1)*

Optimal



Storage:

- Client: $O(K)$
- Server: $O(|DB|)$

Open problem: can we design a completely optimal FP scheme?

Do we have to pay for security?

The future of forward privacy

Many open problems:

- Can we design a completely optimal FP scheme?
- Can we get rid of PK crypto and still be optimal in computation and communication?

Again, what is the cost of security?

Locality of forward privacy

- We can build inefficient FP schemes with low locality: rebuild the DB at every update.
- [DP'17]: FP scheme with $O(\log N)$ update complexity, $O(L)$ locality, $O(N^{1/s}/L)$ read eff. and $O(N.s)$ storage.
- Can we do better?
  Conjecture: optimal updates imply linear locality.
  Intuition: entries with same keyword cannot be 'close'.

Deletions

How to delete entries in an encrypted database?

- Existing schemes use a ‘revocation list’
- Pb: the deleted information is still revealed to the server
- Backward privacy: ‘nothing’ is leaked about the deleted documents

Backward privacy

Brice Minaud
RHUL

Olga Ohrimenko
MSR Cambridge

Backward privacy

Baseline: the client fetches the encrypted lists of inserted and deleted documents, locally decrypts and retrieves the documents.

- ✓ Optimal security
- ✗ 2 interactions
- ✗ $O(a_w)$ communication complexity

Backward privacy with optimal updates & comm.

Could we prevent the server from decrypting some entries?

- Puncturable Encryption [GM'15]: Revocation of decryption capabilities for specific messages
- Encrypt a message with a tag. Revoke the ability to decrypt a set of tags: puncture the secret key
- Based on non-monotonic ABE [OSW'07]

Backward privacy from PE

- Insert *(w, ind)*: encrypt *(w, ind)* with tag $t = H(w,ind)$, and add it to a (possibly FP) SE scheme Σ
- Delete: puncture the secret key on tag $t = H(w,ind)$
- Search *w*: search *w* in Σ and give the punctured SK to the server. Server decrypts the non-deleted results.

Backward privacy from PE

Pb: the punctured SK size grows linearly (# deletions)

- Outsource the storage: put the SK elements in an encrypted DB on the server
- Requires an incremental PE scheme (as [GM'15])
  The puncture alg. only needs a constant fraction of SK

$$Puncture(SK,t) = IncPunct(sk_0,t,d) = (sk'_0, sk_d)$$

- sk_0 is stored locally

Backward privacy from PE

Good:

- Forward & Backward private
- Optimal communication
- Optimal updates

Not so good:

- $O(K)$ client storage
- $O(n_w.d_w)$ search comp.
- Uses pairings (not fast)

Is it possible to do better?
What is this optimal tradeoff?

Verifiable SE

- The server might be malicious: return fake results, delete real results, …
- The client needs to verify the results

David Pointcheval
ENS

Pierre-Alain Fouque
U. Rennes 1

Verifiable SE

This is not free: lower bound (derived from [DNRV'09])

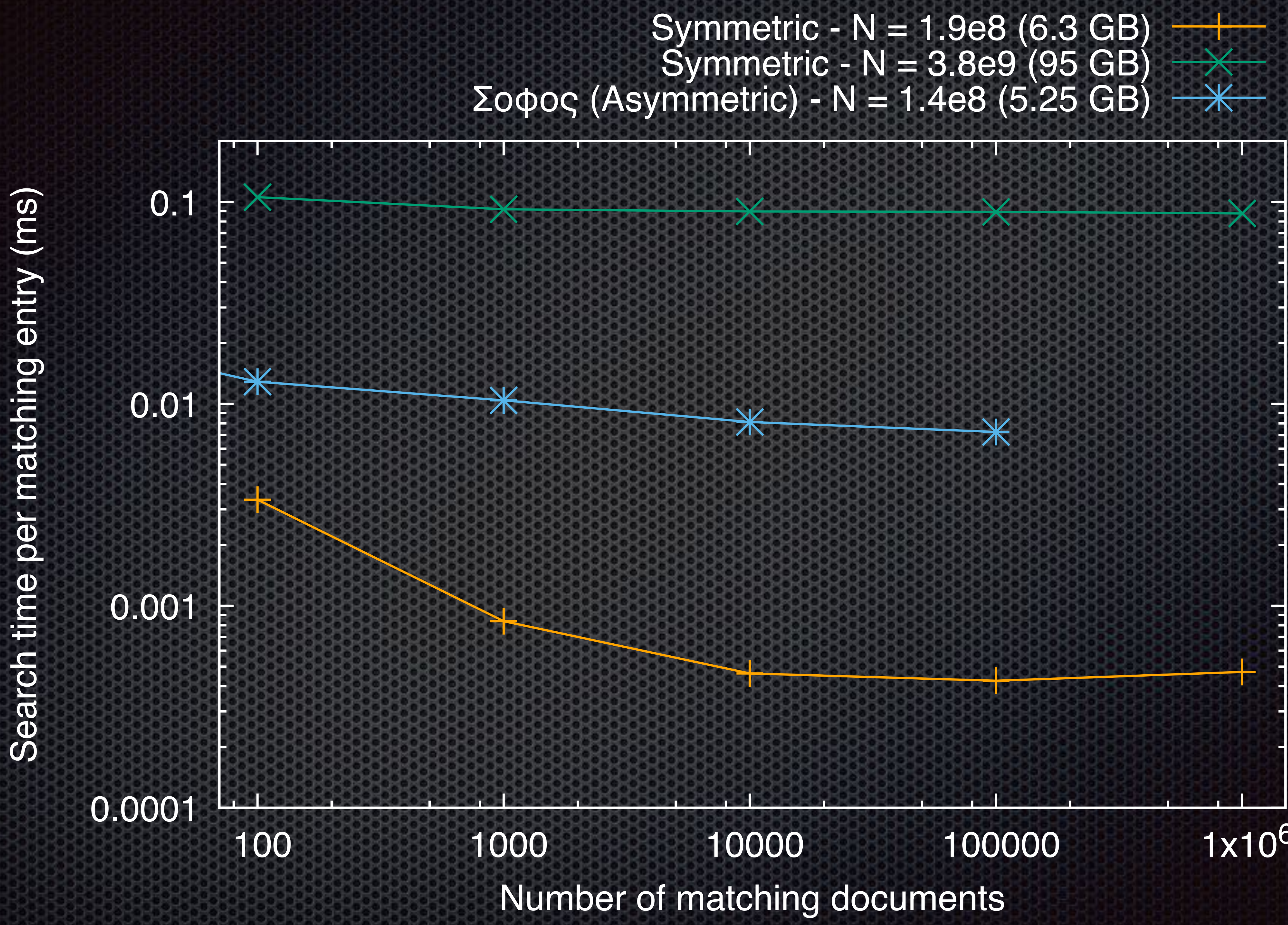
- If client storage is less than $|W|^{1-\varepsilon}$, search complexity has to be larger than $\log |W|$
- The lower bound is tight: using Merkle hash trees and set hash functions
- Many possible tradeoffs between search & update complexities

SE in practice

- In theory, there is no difference between theory and practice…
- Many, many side effects, unexpected behavior, etc, can happen
  - Security: leakage-abuse attacks
  - Implementation details have an impact on efficiency and security

Locality vs. Caching

- The OS is ‘smart’: it caches memory.
- Be careful when you are testing your construction on small databases
- Once the database is cached, non locality disappears
- Beware of the evaluation of performance


Symmetric - N = 1.9e8 (6.3 GB)
Symmetric - N = 3.8e9 (95 GB)
Σοφος (Asymmetric) - N = 1.4e8 (5.25 GB)
Search time per matching entry (ms)
0.1
0.01
0.001
0.0001
100
1000
10000
100000
1x10^6
Number of matching documents

Crypto vs. Seek time

The magic world of searchable encryption:

- Symmetric crypto is free
- Asymmetric crypto is not overly expensive
- A lot of the cost comes from the non-locality of memory accesses

Not-so-snapshot adversary

- Many encrypted databases (CryptDB, ARX, Seabed, CipherCloud, ...) claim security against snapshot adversaries
- Data structures are not history-independent.
  A snapshot leaks about previous operations.
- Snapshot attacks do not take this into account

Today

- Existing implementation of legacy-compatible EDB. Not great security guarantees
- Existing research implementations of index-based SE Clusion (Java), my work (C/C++)
- It would require quite some work to have a production-level implementation of those schemes

Conclusion

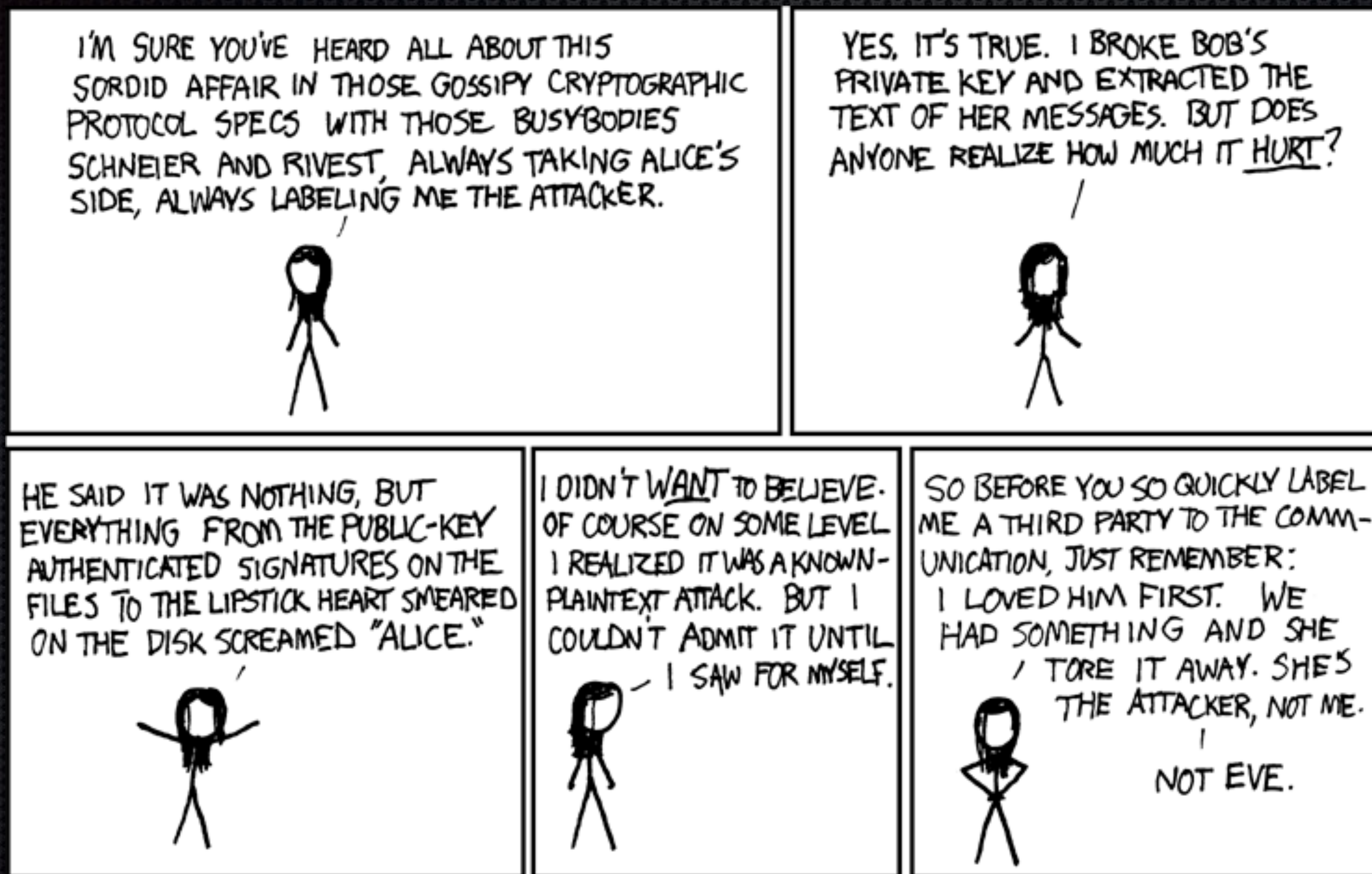
- SE involves very diverse topics: theoretical CS, cryptanalysis, cryptographic primitives, systems, …
- Many open problems (e.g. lower bounds)
- Real world cryptography, with great impact

Bibliography

- SoK: Cryptographically Protected Database Search
  Fuller *et al.* in SP 2017
- See https://r.bost.fyi/se_references/

Slides on my webpage

Questions?


I'M SURE YOU'VE HEARD ALL ABOUT THIS SORDID AFFAIR IN THOSE GOSSIPY CRYPTOGRAPHIC PROTOCOL SPECS WITH THOSE BUSYBODIES SCHNEIER AND RIVEST, ALWAYS TAKING ALICE'S SIDE, ALWAYS LABELING ME THE ATTACKER.
YES, IT'S TRUE. I BROKE BOB'S PRIVATE KEY AND EXTRACTED THE TEXT OF HER MESSAGES. BUT DOES ANYONE REALIZE HOW MUCH IT HURT?
HE SAID IT WAS NOTHING, BUT EVERYTHING FROM THE PUBLIC-KEY AUTHENTICATED SIGNATURES ON THE FILES TO THE LIPSTICK HEART SMEARED ON THE DISK SCREAMED "ALICE."
I DIDN'T WANT TO BELIEVE. OF COURSE ON SOME LEVEL I REALIZED IT WAS A KNOWN-PLAINTEXT ATTACK. BUT I COULDN'T ADMIT IT UNTIL I SAW FOR MYSELF.
SO BEFORE YOU SO QUICKLY LABEL ME A THIRD PARTY TO THE COMMUNICATION, JUST REMEMBER: I LOVED HIM FIRST. WE HAD SOMETHING AND SHE TORE IT AWAY. SHE'S THE ATTACKER, NOT ME.
NOT EVE.