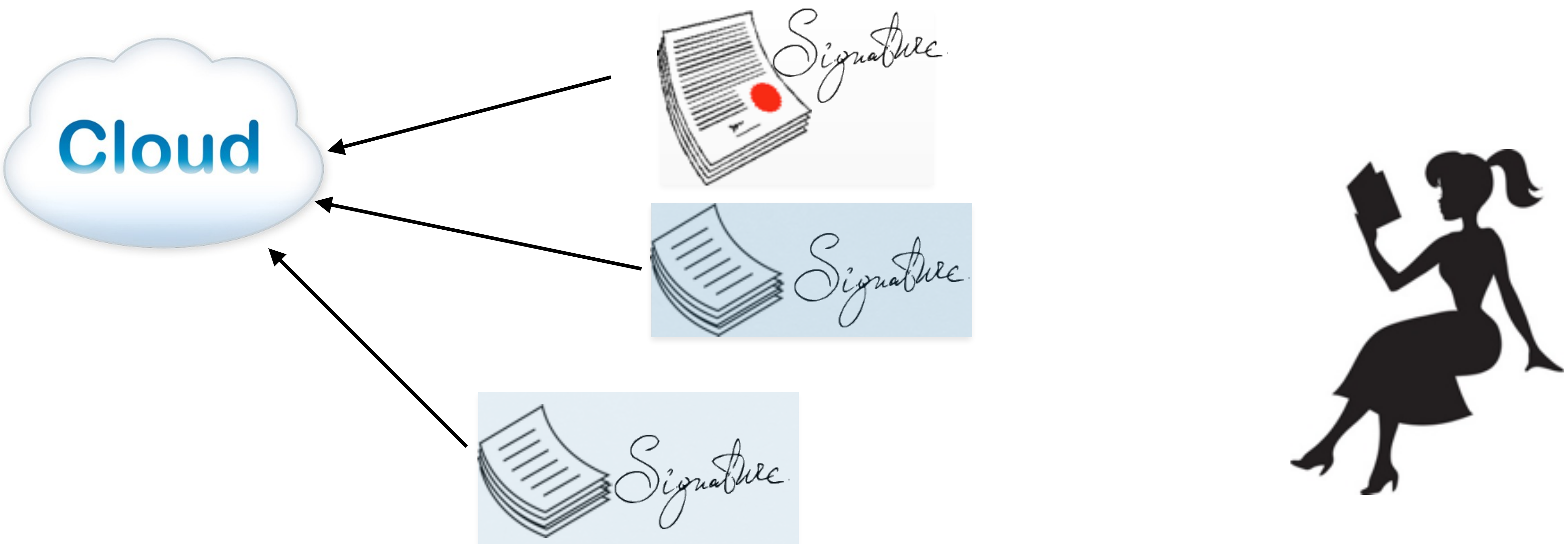
Verifiable Pattern Matching on Outsourced Texts

D. Catalano M. Di Raimondo S. Faro
Università di Catania

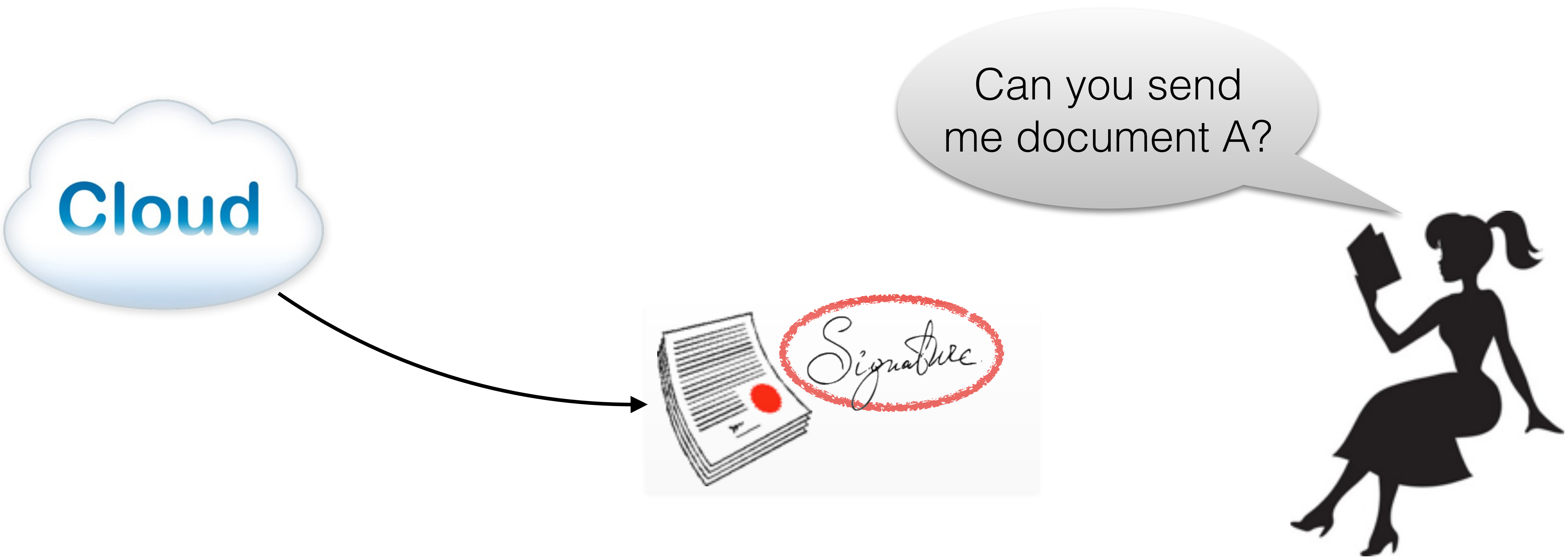
Pattern Matching on Outsourced Documents



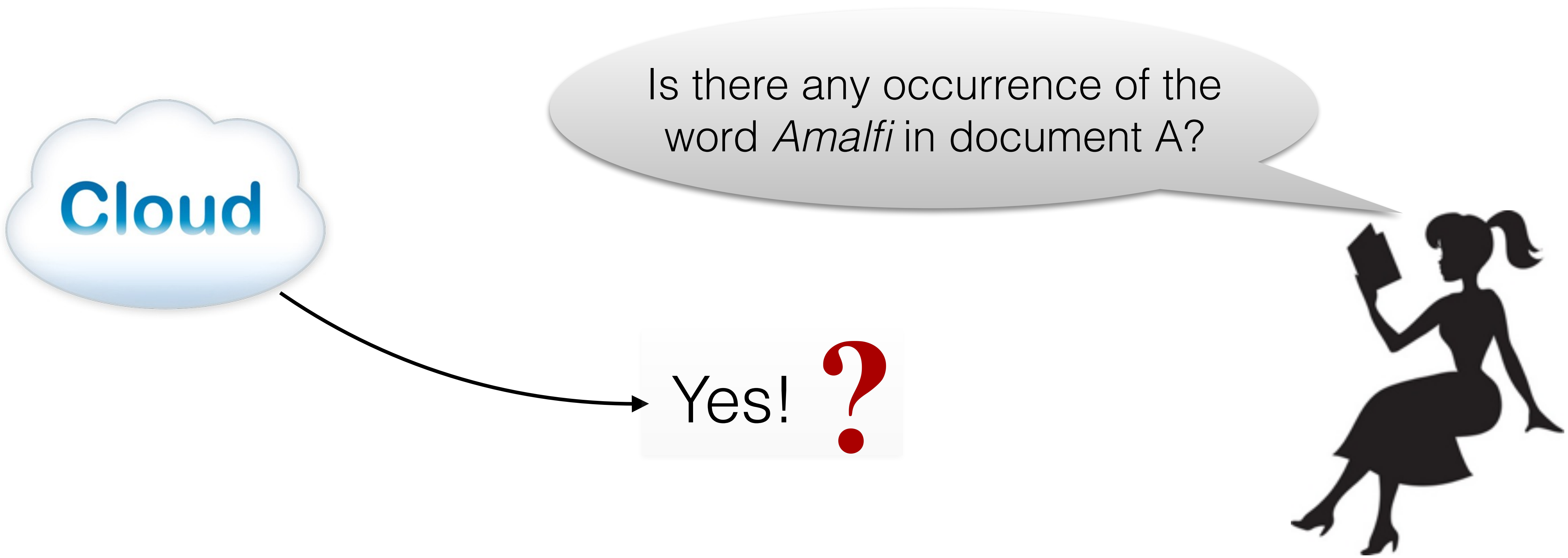
Setting

- Server provides seemingly unbounded storage
- Client has limited storage capabilities (she “forgets” about her data)

Pattern Matching on Outsourced Documents



Pattern Matching on Outsourced Documents



Answers should be

- (Provably) Correct
- Proof of Correctness should be short and easy to check
- Overall workload for the client should be low

Potential Solutions

- AD-SNARKs [BBFR15]
    - Compact ✔
    - Fast Verification ✔
    - Simple and efficient to implement ✖
        - complex machinery, evaluation/verification keys grow (significantly) with the size of the circuit
- (Leveled) Fully Homomorphic Signatures [GVW15] + (any) Pattern Matching algorithm
    - Compact ✔
    - Fast Verification ✔
    - Simple and Efficient to implement ✖

Potential Solutions - II

- Suffix Trees + Cryptographic Accumulators [PPTT15]
  - Compact ✔
  - Fast Verification ✔
  - Simple and Efficient to implement ✔

However
- Significant preprocessing (Client side) is required for *each* document outsourced
- Modifications require redoing preprocessing

Our Solution

Simple and efficient solution based on homomorphic MACs [CF13]

The good 😀

- Compact ✔
- Fast Verification ✔
- Simple and Efficient to implement ✔

The bad ☹️

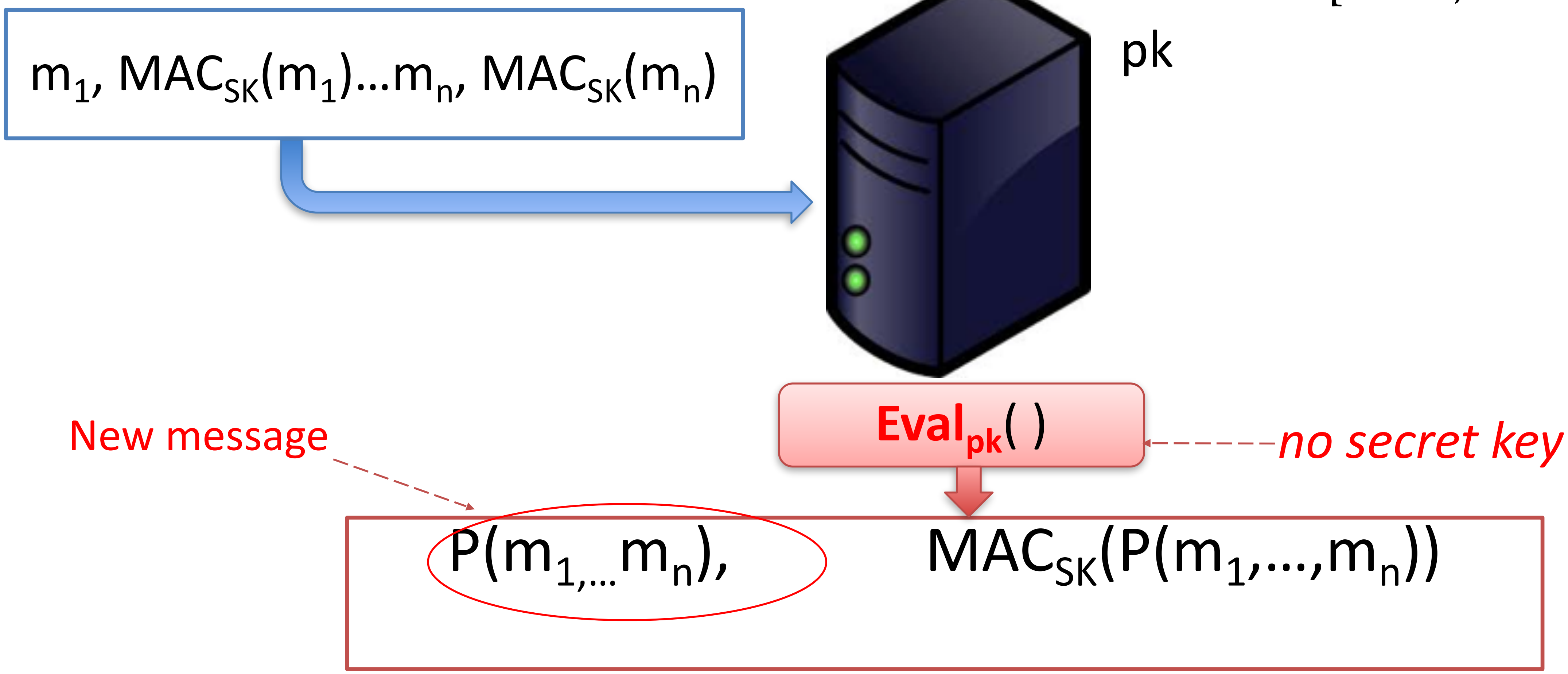
- Practical performances (at server side) only for small texts

Our solution - II

- We develop new pattern matching algorithms that cope well with the fast HoMAC from [CF13]
- Our methods allow to represent *several* text processing operations via low degree polynomials
  - exact/approximate matches,
  - number of (exact/approximate) occurrences,
  - positions of occurrences.
- Very easy to implement.

Interlude: Homomorphic MAC

[AB09, GW13, CF13]



- **Ver**(sk, P, m, σ): Verification w.r.t. $P(m_1,\dots,m_n)$
- **Ver**(sk, P, m, σ) does not know $m_1,\dots,m_n$.
- The actual definition is more complicate

Key Properties

- **Composability:**
  - Outputs of past computations can be used as input for new ones
- **Succinctness:** $|MAC_{SK}(P(m_1,\ldots,m_n))| << |D|$
  - Otherwise trivial solution: send the full (authenticated) D

The Homomorphic MAC [CF13]

MAC(sk, (τ, m)) sk=(k,x)

$r \leftarrow f_k(\tau)$

$y_0 \leftarrow m$

$y_1 \leftarrow (r-m)/x \bmod p$

Return $\sigma = (y_0, y_1)$

Ver(sk, τ, (y_0, y_1),m)

If $(y_0 \neq m)$ return 0

$r \leftarrow f_k(\tau)$

If $(r == xy_1 + y_0)$ return 1

else return 0

- (y_0, y_1) define a linear polynomial $t(z) = y_0 + y_1 z$
- Addition: addition of polynomials
- Multiplication: compute product polynomial (via convolution)
- Very efficient!

String Matching via (low degree) Polynomials

- x,w (binary) patterns, $|x|=|w|=m$

$$x = w \Leftrightarrow \prod_{i=0}^{m-1} (2x_i w_i + 1 - x_i - w_i) = 1$$

- x pattern, $|x|=m$

- y (binary) text, $|y|=n$

Number of occurrences of x in y :

$$\alpha(x,y) = \sum_{j=0}^{n-m} \left(\prod_{i=0}^{m-1} \left(2x_i y_{(j,i)} + 1 - x_i - y_{(j,i)}\right) \right)$$

Proposed protocol

- Client sends out a pattern x (together with its MAC)
- Server homomorphically computes $\boldsymbol{\alpha}$(x,y)

Problem:
- this requires (n-m) computations of 2m-degree polynomials
- very inefficient for large texts

Dynamic Polynomials

- A more careful encoding of the computation can drastically improve performances

- For a given pattern x the computation can be dynamically “adapted”to x

Example

$$\alpha(x,y) = \sum_{j=0}^{n-m} \left(\prod_{i=0}^{m-1} \left(2x_i y_{(j,i)} + 1 - x_i - y_{(j,i)}\right) \right)$$

can be rewritten as

$$\alpha(x,y) = \sum_{j=0}^{n-m} \left(\prod_{i=0}^{m-1} \left(x_i y_{(j,i)} + (1 - x_i)(1 - y_{(j,i)})\right) \right)$$

Dynamic Polynomials - II

$$\alpha(x,y) = \sum_{j=0}^{n-m} \left(\underbrace{\prod_{i=0}^{m-1} \left(x_i y_{(j,i)} + (1-x_i)(1-y_{(j,i)}) \right)} \right)$$

- Knowing the pattern, this can be computed, more efficiently, as

```
P=1
for i=1 to m-1
  if (xi=0) P=P × (1-y(j,i))
  else P=P × y(j,i)
```

This alone reduces the computational costs of the server by a (rough) 70%

Experiments

- 4 char pattern
  - 10 KiB text

Proof Size	Evaluation	Verification
528 bytes	4 s	300 ms

  - 100 KiB text

Proof Size	Evaluation	Verification
528 bytes	38 s	3 s

Experiments - II

- 8 char pattern
  - 10 KiB text

Proof Size	Evaluation	Verification
1040 bytes	15 s	1 s

  - 100 KiB text

Proof Size	Evaluation	Verification
1040 bytes	151 s	6 s

Conclusions and Open Questions

- We considered the question of performing pattern matching reliably on outsourced documents.
- Our solutions are reasonably efficient but not yet practical.
- Can we come up with better (i.e. more efficient) homomorphic authenticators?

Thank you!