Outsourced Pattern Matching

Daniele Venturi

40th International Colloquium on Automata, Languages and Programming (ICALP 2013)
Latvia, Riga 09-07-2013

Joint work with Sebastian Faust and Carmit Hazay
Delegatable computation

- Outsourcing computation of a public function $f$ to a possibly malicious cloud provider (a.k.a. the server)
Delegatable computation

- Outsourcing computation of a public function $f$ to a possibly malicious cloud provider (a.k.a. the server)

Weak client

input $x$
Delegatable computation

- **Outsourcing** computation of a **public** function $f$ to a possibly malicious cloud provider (a.k.a. the server)

Weak client

Cloud server

input $x$
Delegatable computation

- **Outsourcing** computation of a **public** function $f$ to a possibly **malicious** cloud provider (a.k.a. the server)

Weak client

```
input $x$
```

Cloud server

```
“Compute $f(x)$ for me!”
```

Desirable goals:
- Verify correctness much easier than computing $f(x)$ from scratch
- Minimize communication complexity

Two main lines of work:
- Protocols for any function (e.g. [GGP10])
- Protocols for a specific function (e.g. [BGV11])

This talk!

D. Venturi (Aarhus University)
Delegatable computation

- **Outsourcing** computation of a **public** function $f$ to a possibly **malicious** cloud provider (a.k.a. the server)

  Weak client

  input $x$

  Cloud server

  "Compute $f(x)$ for me!"

  Is my data private? Is the result correct?

Desirable goals:

- Verify correctness much easier than computing $f(x)$ from scratch
- Minimize communication complexity

Two main lines of work:

- Protocols for any function (e.g. [GGP10])
- Protocols for a specific function (e.g. [BGV11])

This talk!

D. Venturi (Aarhus University)
Delegatable computation

- **Outsourcing** computation of a **public** function $f$ to a possibly **malicious** cloud provider (a.k.a. the server)

Weak client

"Compute $f(x)$ for me!"

Cloud server

Is my data private?

Is the result correct?

Desirable goals:

- Verify correctness much easier than computing $f(x)$ from scratch
- Minimize communication complexity

Two main lines of work:

- Protocols for any function (e.g. [GGP10])
- Protocols for a specific function (e.g. [BGV11])

This talk!
Delegatable computation

- **Outsourcing** computation of a public function $f$ to a possibly malicious cloud provider (a.k.a. the server)

Weak client

Cloud server

Input $x$

Is my data private?
Is the result correct?

Desirable goals:
- Verify correctness much easier than computing $f(x)$ from scratch
Delegatable computation

- Outsourcing computation of a public function \( f \) to a possibly malicious cloud provider (a.k.a. the server)

Weak client

Cloud server

"Compute \( f(x) \) for me!"

Is my data private?
Is the result correct?

Desirable goals:
- Verify correctness much easier than computing \( f(x) \) from scratch
- Minimize communication complexity
Delegatable computation

- Outsourcing computation of a public function $f$ to a possibly malicious cloud provider (a.k.a. the server)

Desirable goals:
- Verify correctness much easier than computing $f(x)$ from scratch
- Minimize communication complexity
- Two main lines of work:
Delegatable computation

- **Outsourcing** computation of a **public** function $f$ to a possibly **malicious** cloud provider (a.k.a. the server)

Desirable goals:
- Verify correctness **much easier** than computing $f(x)$ from scratch
- Minimize communication complexity

Two main lines of work:
- Protocols for **any** function (e.g. [GGP10])

Is my data private? Is the result correct?
Delegatable computation

- **Outsourcing** computation of a public function $f$ to a possibly malicious cloud provider (a.k.a. the server)

Weak client

Cloud server

```
input $x$
```

Is my data private?
Is the result correct?

Desirable goals:
- Verify correctness **much easier** than computing $f(x)$ from scratch
- Minimize communication complexity

Two main lines of work:
- Protocols for any function (e.g. [GGP10])
- Protocols for a **specific** function (e.g. [BGV11])

Delegatable computation

- **Outsourcing** computation of a **public** function $f$ to a possibly malicious cloud provider (a.k.a. the server)

![Diagram showing weak client and cloud server, with input $x$ and questions: Is my data private? Is the result correct?]

- **Desirable goals:**
  - Verify correctness **much easier** than computing $f(x)$ from scratch
  - **Minimize** communication complexity
- **Two main lines of work:**
  - Protocols for **any** function (e.g. [GGP10])
  - Protocols for a **specific** function (e.g. [BGV11])

**This talk!**
Pattern matching

**Input:** A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
Pattern matching

- **Input**: A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output**: The set of positions where $p$ appears in $T$ (if any)
Pattern matching

- **Input**: A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output**: The set of positions where $p$ appears in $T$ (if any)
Pattern matching

- **Input**: A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output**: The set of positions where $p$ appears in $T$ (if any)

Protocol $\pi_{pm}$

$$T \in \Sigma^n \quad \text{protocol} \quad \pi_{pm} \quad p \in \Sigma^m$$
Pattern matching

**Input:** A text \( T \in \Sigma^n \) and a pattern \( p \in \Sigma^m \) (e.g., \( \Sigma = \{0, 1\} \))

**Output:** The set of positions where \( p \) appears in \( T \) (if any)
Pattern matching

**Input:** A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)

**Output:** The set of positions where $p$ appears in $T$ (if any)

- Doesn’t learn anything and doesn’t learn anything beyond $(i_1, \ldots, i_t)$

---

protocol $\pi_{pm}$

$T \in \Sigma^n$

$p \in \Sigma^m$

$(i_1, \ldots, i_t)$
Pattern matching

- **Input:** A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output:** The set of positions where $p$ appears in $T$ (if any)

- protocol $\pi_{pm}$

- $T \in \Sigma^n$
- $p \in \Sigma^m$
- $(i_1, \ldots, i_t)$

- doesn’t learn anything and doesn’t learn anything beyond $(i_1, \ldots, i_t)$

- Broad set of applications: text retrieval, music retrieval, computational biology, data mining, network security...
Pattern matching

- **Input:** A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output:** The set of positions where $p$ appears in $T$ (if any)

Protocol $\pi_{pm}$

$T \in \Sigma^n$ \hspace{1cm} \text{protocol} \hspace{1cm} p \in \Sigma^m$

\[(i_1, \ldots, i_t)\]

- Doesn’t lean anything and doesn’t learn anything beyond $(i_1, \ldots, i_t)$
- Broad set of applications: text retrieval, music retrieval, computational biology, data mining, network security...
- Solutions for the 2-party case **not applicable** to the cloud setting
Pattern matching

- **Input**: A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output**: The set of positions where $p$ appears in $T$ (if any)

Protocol $\pi_{pm}$

- doesn’t lean anything and doesn’t learn anything beyond $(i_1, \ldots, i_t)$
- Broad set of applications: text retrieval, music retrieval, computational biology, data mining, network security...
- Solutions for the 2-party case not applicable to the cloud setting
  - Overhead per search query grows linearly in $n$
Pattern matching

- **Input**: A text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$ (e.g., $\Sigma = \{0, 1\}$)
- **Output**: The set of positions where $p$ appears in $T$ (if any)

Protocol $\pi_{pm}$ doesn’t learn anything and doesn’t learn anything beyond $(i_1, \ldots, i_t)$

Broad set of applications: text retrieval, music retrieval, computational biology, data mining, network security...

Solutions for the 2-party case not applicable to the cloud setting
- Overhead per search query grows linearly in $n$
- Text holder cannot control the content of the server’s responses
Setup phase: outsources an encoding $\tilde{T}$ of $T$ to $\pi$.
Pattern matching in the cloud

1. Setup phase: outsources an encoding $\tilde{T}$ of $T$ to

$$T \in \Sigma^n$$
Pattern matching in the cloud

1. **Setup phase:** outsources an encoding \( \tilde{T} \) of \( T \) to

\[
T \in \Sigma^n
\]
Pattern matching in the cloud

1 Setup phase: outsources an encoding $\tilde{T}$ of $T$ to $\tilde{T} \in \Sigma^\ell$

$T \in \Sigma^n$
Pattern matching in the cloud

1. Setup phase: outsources an encoding $\tilde{T}$ of $T$ to

$\tilde{T} \in \Sigma^\ell$

$T \in \Sigma^n$

2. Query phase: interacts with $p$ interacts with $\tilde{T}$ to get $(i_1, \ldots, i_t)$
Pattern matching in the cloud

1. Setup phase: outsources an encoding $\tilde{T}$ of $T$ to

\[ \tilde{T} \in \Sigma^\ell \]

\[ T \in \Sigma^n \]

2. Query phase: with $p$ interacts with to get $(i_1, \ldots, i_t)$
Pattern matching in the cloud

1. Setup phase: outsources an encoding $\tilde{T}$ of $T$ to $\pi$.

2. Query phase: with $p$ interacts with to get $(i_1, \ldots, i_t)$.
Pattern matching in the cloud

1. Setup phase: outsources an encoding $\tilde{T}$ of $T$ to

$$\tilde{T} \in \Sigma^\ell$$

2. Query phase: with $p$ interacts with to get $(i_1, \ldots, i_t)$
   - To avoid disclosure of too much information about $T$ we need to “bind” a search query to $p$
Pattern matching in the cloud

1. **Setup phase:** outsources an encoding $\tilde{T}$ of $T$ to

\[
\tilde{T} \in \Sigma^\ell
\]

2. **Query phase:** with $p$ interacts with to get $(i_1, \ldots, i_t)$

- To avoid disclosure of too much information about $T$ we need to “bind” a search query to $p$
- We do so by letting interact with before running $\pi_{Opm}$
Pattern matching in the cloud

1. **Setup phase:** outsources an encoding $\tilde{T}$ of $T$ to

   \[\tilde{T} \in \Sigma^\ell\]

   we can think of $R_p$ as a trapdoor allowing to search $p$ in $\tilde{T}$

2. **Query phase:** $p$ interacts with $\tilde{T}$ to get $(i_1, \ldots, i_t)$

   - To avoid disclosure of too much information about $T$ we need to "bind" a search query to $p$
   - We do so by letting $\tilde{T}$ interact with $\pi_{\text{Pre}}$ before running $\pi_{\text{Query}}$
Pattern matching in the cloud

1. Setup phase: outsources an encoding $\tilde{T}$ of $T$ to

$$\tilde{T} \in \Sigma^\ell$$

we can think of $R_p$ as a trapdoor allowing to search $p$ in $\tilde{T}$

$$\pi_{\text{Pre}} \quad \pi_{\text{Query}} \quad \pi_{\text{Opm}}$$

2. Query phase: with $p$ interacts with to get $(i_1, \ldots, i_t)$

- To avoid disclosure of too much information about $T$ we need to “bind” a search query to $p$
- We do so by letting interact with before running $\pi_{\text{Opm}}$
Summary of results

- A precise \textit{(simulation-based)} security definition for outsourced pattern matching
Summary of results

- A precise (simulation-based) security definition for outsourced pattern matching
- A simple protocol $(\pi_{Pre}, \pi_{Query}, \pi_{Opm})$ with passive security based on the subset sum problem

Amortized complexity: Communication/computation linear in $n$ for $\pi_{Pre}$, but linear in $m$ during $\pi_{Opm}$ (optimal)

The server is allowed to learn the matched positions for each query (this seems necessary if we want sublinear communication in the query phase)

An extension achieving active security (see the paper)
Summary of results

- A precise (simulation-based) security definition for outsourced pattern matching
- A simple protocol \( (\pi_{\text{Pre}}, \pi_{\text{Query}}, \pi_{\text{Opm}}) \) with passive security based on the subset sum problem
  - Amortized complexity: Communication/computation linear in \( n \) for \( \pi_{\text{Pre}} \), but linear in \( m \) during \( \pi_{\text{Opm}} \) (optimal)
Summary of results

- A precise (simulation-based) security definition for outsourced pattern matching
- A simple protocol \((\pi_\text{Pre}, \pi_\text{Query}, \pi_\text{Opm})\) with passive security based on the subset sum problem
  - Amortized complexity: Communication/computation linear in \(n\) for \(\pi_\text{Pre}\), but linear in \(m\) during \(\pi_\text{Opm}\) (optimal)
  - The server is allowed to learn the matched positions for each query (this seems necessary if we want sublinear communication in the query phase)
Summary of results

- A precise (simulation-based) security definition for outsourced pattern matching
- A simple protocol \((\pi_{\text{Pre}}, \pi_{\text{Query}}, \pi_{\text{Opm}})\) with passive security based on the subset sum problem
  - Amortized complexity: Communication/computation linear in \(n\) for \(\pi_{\text{Pre}}\), but linear in \(m\) during \(\pi_{\text{Opm}}\) (optimal)
  - The server is allowed to learn the matched positions for each query (this seems necessary if we want sublinear communication in the query phase)
- An extension achieving active security (see the paper)
Roadmap

- Security Definition
- Active Security
- Passive Security
- Construction
- Proof Idea

Rest of this Talk
Roadmap

Rest of this Talk

Security Definition

Passive Security

Construction

Active Security

Proof Idea
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm
We define security following the real/ideal world paradigm.
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm

\[ T \in \Sigma^n \]

**ideal world**

**real world**

\[ \forall, \exists \text{ s.t. REAL}(\pi, \omega) \approx IDEAL(\omega, \omega) \]
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm

\[ T \in \Sigma^n \]

\[ n, m \]

Ideal world
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm

\[ T \in \Sigma^n \]

\[ p \in \Sigma^m \]

\[ \text{ideal world} \]

\[ n, m \]
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm

```
\text{ideal world}
```

```
T \in \Sigma^n
```

```
p \in \Sigma^m
```

```
(i_1, \ldots, i_t)
```

```
\text{allow?}
```

```
\text{OK}
```

```
\text{allow}
```

```
\forall, \exists \text{ s.t. } \text{REAL}(\pi) \approx \text{IDEAL}(\pi)
```

```
D. Venturi (Aarhus University)
```

```
Outsourced Pattern Matching
```

```
ICALP2013@Riga
```

```
8 / 19
```
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm.
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm

**real world**

\[ \tilde{T}, m, (i_1, \ldots, i_t) \]

\[ T \in \Sigma^n \]

\[ p \in \Sigma^m \]

\[ R_p, (i_1, \ldots, i_t) \]

**ideal world**

\[ n, m, (i_1, \ldots, i_t) \]

\[ T \in \Sigma^n \]

\[ \text{"allow"} \]

\[ (i_1, \ldots, i_t) \]

\[ p \]

\[ p \in \Sigma^m \]
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm

---

**real world**

\[ \tilde{T}, m, (i_1, \ldots, i_t) \]

**ideal world**

\[ n, m, (i_1, \ldots, i_t) \]

\[ T \in \Sigma^n \]

\[ p \in \Sigma^m \]

\[ R_p, (i_1, \ldots, i_t) \]

\[ T \in \Sigma^n \]

\[ \text{"allow?"} \]

\[ \text{"OK"} \]

\[ \text{"allow?"} \]
Security of outsourced pattern matching

- We define security following the real/ideal world paradigm
Technical challenges in the simulation

- Let’s look at the case of a passively corrupted server
Technical challenges in the simulation

- Let’s look at the case of a passively corrupted server

\[ T \in \Sigma^n \]
Technical challenges in the simulation

- Let’s look at the case of a **passively** corrupted server
Technical challenges in the simulation

- Let’s look at the case of a **passively** corrupted server

- T ∈ Σ^n has a hard life:

  - πPre

  - T ∈ Σ^n

  - has a hard life:
Technical challenges in the simulation

Let’s look at the case of a **passively** corrupted server

\[ \tilde{T}' \in \Sigma^n \]

It has a hard life:

- It has to simulate \( \tilde{T}' \) as output of \( \pi_{\text{Pre}} \).
Technical challenges in the simulation

Let’s look at the case of a **passively** corrupted server

\[ T \in \Sigma^n \]

\[ \tilde{T} \]

\[ \tilde{T}' \]

\[ \tilde{T}' \]

\[ \tilde{T} \]

\[ p \in \Sigma^m \]

\[ \pi_{\text{Pre}} \]

\[ \pi_{\text{Query}} \]

has a hard life:

- It has to simulate \( \tilde{T}' \) as output of \( \pi_{\text{Pre}} \)
- When later \( p \) is searched, it gets to know \( (i_1, \ldots, i_t) \) and has to cook up a consistent \( R'_p \) such that

\[
(\tilde{T}', R'_p) \approx (\tilde{T}, R_p)
\]
Technical challenges in the simulation

- Let’s look at the case of a **passively** corrupted server

Diagram:

- \( \tilde{T} \in \Sigma^n \)
- \( \pi_{Pre} \)
- \( \pi_{Query} \)
- \( R' \)
- \( (i_1, \ldots, i_t) \)
- \( p \in \Sigma^m \)

- **\( T \)** has a hard life:
  - It has to simulate \( \tilde{T}' \) as output of \( \pi_{Pre} \)
  - When later \( p \) is searched, it gets to know \( (i_1, \ldots, i_t) \) and has to cook up a **consistent** \( R'_p \) such that

\[
(\tilde{T}', R'_p) \approx (\tilde{T}, R_p)
\]
Technical challenges in the simulation

- Let’s look at the case of a **passively** corrupted server

Let’s consider the situation where a server is passively corrupted.

### Technical Challenges

- The server, $T \in \Sigma^n$, has a hard life:
  - It has to simulate $\tilde{T}'$ as output of $\pi_{Pre}$.
  - When later $p$ is searched, it gets to know $(i_1, \ldots, i_t)$ and has to cook up a consistent $R'_p$ such that
    $$ (\tilde{T}', R'_p) \approx (\tilde{T}, R_p) $$

However, $p$ was not known when $\tilde{T}'$ has been computed!
Subset sum

- Let $\ell$ and $M$ be integers
Subset sum

- Let \( \ell \) and \( M \) be integers

\[
\begin{array}{c}
a_1 \ldots a_\ell \\
\in \mathbb{Z}_M^\ell
\end{array}
\]
Subset sum

- Let $\ell$ and $M$ be integers

Let $a_1, \ldots, a_\ell \in \mathbb{Z}_M^\ell$ and $s_1, \ldots, s_\ell \in \{0,1\}$ such that

\[
s_i \cdot a_i \mod M = R_i
\]

for $i = 1, \ldots, \ell$. Find $(s_1, \ldots, s_\ell)$.

Observation: For random $s'$, the probability that a random $s'$ shares the same $R$ with $s$ is at most $2\ell / M$.

Hardness of subset sum as a function of $\Delta = \ell / \log M$.

\[
\frac{\ell}{\log 2} \leq \Delta \leq \frac{\ell}{\log 2\ell}
\]

easy

hard
Subset sum

- Let $\ell$ and $M$ be integers

\[
\begin{align*}
\underbrace{a_1 \ldots a_{\ell}}_{\in \mathbb{Z}_M^\ell} & \cdot \\
\underbrace{s_1 \ldots s_{\ell}}_{\in \{0,1\}} & \mod M = R
\end{align*}
\]
Let $\ell$ and $M$ be integers

\[
a_1 \ldots a_\ell \equiv s_1 \ldots s_\ell \pmod{M} = R
\]

\[
\in \mathbb{Z}_M^\ell
\]

\[
\in \{0,1\}
\]

Goal

given $(R, a_1, \ldots, a_\ell)$, find $(s_1, \ldots, s_\ell)$

$s_i = 1$ means $a_i$ contributes to the summation

Observation: For random $s$, the probability that a random $s'$ shares the same $R$ with $s$ is $\leq \frac{2\ell}{M}$
Subset sum

Let $\ell$ and $M$ be integers

\[
\begin{align*}
    a_1 & \quad \ldots \quad a_\ell \\
\in & \quad \mathbb{Z}_M^\ell \\
    s_1 & \quad \vdots \\
\in & \quad \{0,1\} \\
    \vdots & \\
    s_\ell & \\
\end{align*}
\mod M = R
\]

**Goal**

given $(R, a_1, \ldots, a_\ell)$, find $(s_1, \ldots, s_\ell)$

$s_i = 1$ means $a_i$ contributes to the summation

**Observation**: For random $s$, $a$ the probability that a random $s'$ shares the same $R$ with $s$ is $\leq 2^\ell/M$
Subset sum

- Let $\ell$ and $M$ be integers

$$a_1 \ldots a_\ell \mod{M} = R$$

Given $(R, a_1, \ldots, a_\ell)$, find $(s_1, \ldots, s_\ell)$

$s_i = 1$ means $a_i$ contributes to the summation

Observation: For random $s$, $a$ the probability that a random $s'$ shares the same $R$ with $s$ is $\leq 2^{\ell}/M$

Hardness of subset sum as a function of $\Delta = \ell/\log M$
Subset sum

- Let $\ell$ and $M$ be integers

$$a_1 \ldots a_\ell \mod M = R$$

$s_i = 1$ means $a_i$ contributes to the summation.

**Observation:** For random $s$, $a$ the probability that a random $s'$ shares the same $R$ with $s$ is $\leq 2^\ell / M$

**Hardness** of subset sum as a function of $\Delta = \ell / \log M$
The basic idea...

- The $\pi_{Pre}$ protocol:
The basic idea...

- The $\pi_{\text{Pre}}$ protocol:

$$T = \begin{array}{cccccc}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}$$

$p \in \{0, 1\}^3$

(t matches)
The basic idea...

- The $\pi_{\text{Pre}}$ protocol:

\[
T = \begin{array}{ccccccc}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

\[p \in \{0,1\}^3 \]

(t matches)

\[
\forall p \subseteq T
\]

choose random $R_p$, $(a_1, \ldots, a_t)$

s.t. $\sum_{i=1}^{t} a_i = R_p \mod M$
The basic idea...

- The $\pi_{\text{Pre}}$ protocol:

\[ T = \begin{array}{cccccc}
0 & 0 & 1 & 1 & 0 & 0 \end{array} \begin{array}{c} 1 \end{array} \begin{array}{c} 1 \end{array} \begin{array}{c} 0 \end{array} \Rightarrow \tilde{T} = a_1 \ldots a_\ell \ (\ell = n - m + 1) \]

\[ p \in \{0,1\}^3 \]

\[ (t \text{ matches}) \]

\[ \forall p \subseteq T \]

choose random $R_p, (a_1, \ldots, a_t)$

s.t. $\sum_{i=1}^{t} a_i = R_p \mod M$
The basic idea...

The $\pi_{\text{Pre}}$ protocol:

$$T = \begin{array}{cccccccc}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0
\end{array} \Rightarrow \tilde{T} = \begin{array}{cccc}
a_1 & \ldots & a_\ell
\end{array} (\ell = n - m + 1)$$

$$p \in \{0,1\}^3 \quad (t \text{ matches})$$

$\forall p \subseteq T$

choose random $R_p$, $(a_1, \ldots, a_t)$

s.t. $\sum_{i=1}^t a_i = R_p \mod M$

(In practice compute $R_p = f(\kappa, p)$ for PRF $f$ and store only $\kappa$.)
The basic idea...

- The $\pi_{\text{Pre}}$ protocol:

\[ T = \underbrace{0 \ 0 \ 1 \ 1 \ 0} \ 0 \ 1 \ 1 \ 0 \Rightarrow \tilde{T} = a_1 \ldots a_\ell \ (\ell = n - m + 1) \]

\[ p \in \{0,1\}^3 \ (t \text{ matches}) \]

\[
\forall p \subseteq T \\
\text{choose random } R_p, \ (a_1, \ldots, a_t) \\
s.t. \ \sum_{i=1}^{t} a_i = R_p \mod M
\]

(In practice compute $R_p = f(\kappa, p)$ for PRF $f$ and store only $\kappa$.)

- Protocol $\pi_{\text{Query}}$: Any two-party protocol for oblivious evaluation of $f(\kappa, p)$
The basic idea...

- The $\pi_{Pre}$ protocol:

$T = 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ \Rightarrow \ \tilde{T} = a_1 \ldots a_\ell$ ($\ell = n - m + 1$)

\[ \forall p \subseteq T \]

choose random $R_p$, $(a_1, \ldots, a_t)$

s.t. \[ \sum_{i=1}^{t} a_i = R_p \ mod \ M \]

(In practice compute $R_p = f(\kappa, p)$ for PRF $f$ and store only $\kappa$.)

- Protocol $\pi_{Query}$: Any two-party protocol for oblivious evaluation of $f(\kappa, p)$

- Protocol $\pi_{Opm}$: gives $R_p$ to \[ \star \] and the latter solves subset sum instance $(R_p, \tilde{T})$
... and its limitation

- The above simple protocol can be proven secure, but suffers from two limitations:

  Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter).

To keep the collision probability ($= 2^\ell / M$) low we shall set $M = 2^{\lambda + n}$. This yields $\ell < \sqrt{\lambda}$ if we want the subset sum problem to be solvable in polynomial time.

Even $\lambda = 10^4$ (i.e., subset sum elements of size $\approx 10$ KByte) allows to process texts of less than 100 bits.

To overcome the above problems, we define an extension of the previous solution based on packaging.
The above simple protocol can be proven secure, but suffers from two limitations:

- Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter)
... and its limitation

- The above simple protocol can be proven secure, but suffers from two limitations:
  - Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter)
  - Limited use, as the text has to be very short
The above simple protocol can be proven secure, but suffers from two limitations:

- Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter)
- **Limited use**, as the text has to be very short
  - To keep the collision probability ($= 2^\ell / M$) low we shall set $M = 2^{\lambda + n}$
... and its limitation

- The above simple protocol can be proven secure, but suffers from two limitations:
  - Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter)
  - Limited use, as the text has to be very short
    - To keep the collision probability ($= 2^\ell / M$) low we shall set $M = 2^{\lambda+n}$
    - This yields $\ell < \sqrt{\lambda}$ if we want the subset sum problem to be solvable in polynomial time
The above simple protocol can be proven secure, but suffers from two limitations:

- Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter)
- Limited use, as the text has to be very short

  - To keep the collision probability ($= 2^{\ell}/M$) low we shall set $M = 2^{\lambda+n}$
  - This yields $\ell < \sqrt{\lambda}$ if we want the subset sum problem to be solvable in polynomial time
  - Even $\lambda = 10^4$ (i.e., subset sum elements of size $\approx 10$ KByte) allows to process texts of less than 100 bits
... and its limitation

- The above simple protocol can be proven secure, but suffers from two limitations:
  - Communication complexity is $O(n^2 + \lambda n)$ in the setup phase and proportional to $n$ in the query phase ($\lambda$ is security parameter)
  - Limited use, as the text has to be very short
    - To keep the collision probability ($= 2^\ell / M$) low we shall set $M = 2^{\lambda+n}$
    - This yields $\ell < \sqrt{\lambda}$ if we want the subset sum problem to be solvable in polynomial time
    - Even $\lambda = 10^4$ (i.e., subset sum elements of size $\approx$ 10 KByte) allows to process texts of less than 100 bits

- To overcome the above problems, we define an extension of the previous solution based on packaging
Packaging

- Divide the original text $T$ into **blocks** of length $2m$ (overlapping in the last $m$ bits)
Packaging

- Divide the original text $T$ into blocks of length $2m$ (overlapping in the last $m$ bits)

$$T = 001100110$$
Packaging

- Divide the original text $T$ into blocks of length $2m$ (overlapping in the last $m$ bits)

$$T = \text{001100110} \quad B_1 = \text{0011}$$

$$B_2 = \text{1100} \quad B_3 = \cdots$$
Packaging

- Divide the original text $T$ into blocks of length $2m$ (overlapping in the last $m$ bits)

$$T = 001100110$$

$$B_1 = 0011$$

$$B_2 = 1100$$

$$B_3 = \cdots$$

- Process each of the blocks separately as before
Packaging

- Divide the original text $T$ into **blocks** of length $2m$ (overlapping in the last $m$ bits)

  \[ T = 001100110 \]

  \[ B_1 = 0011 \]

  \[ B_2 = 1100 \]

  \[ B_3 = \cdots \]

- Process each of the blocks **separately** as before
  - Avoid using in each block the same trapdoor for some pattern $p$
Packaging

- Divide the original text $T$ into blocks of length $2m$ (overlapping in the last $m$ bits)

$$T = 001100110$$

$$B_1 = 0011$$

$$B_2 = 1100$$

$$B_3 = \cdots$$

- Process each of the blocks separately as before
  - Avoid using in each block the same trapdoor for some pattern $p$
  - To do so we encode $R_p = \mathcal{H}(f(\kappa, p)||b)$ where $\mathcal{H}$ is a random oracle and $b$ is the block number
Packaging

- Divide the original text $T$ into blocks of length $2m$ (overlapping in the last $m$ bits)

  $T = 001100110$

  $B_1 = 0011$

  $B_2 = 1100$

  $B_3 = \ldots$

- Process each of the blocks separately as before
  - Avoid using in each block the same trapdoor for some pattern $p$
  - To do so we encode $R_p = \mathcal{H}(f(\kappa, p)||b)$ where $\mathcal{H}$ is a random oracle and $b$ is the block number
Packaging

- Divide the original text $T$ into blocks of length $2m$ (overlapping in the last $m$ bits)

\[
B_1 = 0011 \\
B_2 = 1100 \\
B_3 = \ldots
\]

- Process each of the blocks separately as before
  - Avoid using in each block the same trapdoor for some pattern $p$
  - To do so we encode $R_p = H(f(\kappa, p)||b)$ where $H$ is a random oracle and $b$ is the block number

The communication complexity is $O(mn + \lambda n)$ in the setup phase and $O(\lambda m)$ in the query phase
How the simulator works

- Let’s look again at the case of a **passively** corrupted server
How the simulator works

- Let’s look again at the case of a **passively** corrupted server

\[ T \in \Sigma^n \]
How the simulator works

- Let’s look again at the case of a passively corrupted server
How the simulator works

- Let’s look again at the case of a **passively** corrupted server
How the simulator works

- Let’s look again at the case of a passively corrupted server
How the simulator works

- Let’s look again at the case of a passively corrupted server

\[ \tilde{T}' \leftarrow \mathbb{Z}_M^\ell \]

Sample \( R'_p \) at random; for each \( i_j \) compute the block \( b \) and program \( \mathcal{H}(R'_p||b) := \sum \tilde{T}[\text{matching indexes in } b] \)
How the simulator works

- Let’s look again at the case of a passively corrupted server

Let $T \in \Sigma^n$ be the input pattern. The simulator works by simulating the behavior of the real system. For each query $\pi_{Query}$, the simulator samples $R'_p$ at random; for each $i_j$ compute the block $b$ and program $H(R'_p||b) := \sum \tilde{T}$[matching indexes in $b$]. The simulator then outputs $\tilde{T}'$ as the result, where $\tilde{T}' \leftarrow \mathbb{Z}_M^\ell$. The simulator is defined to be the same as the real distribution but such that $R_p$ is random, $\text{HYB}(\pi) \approx c_{\text{REAL}}(\pi)$ (by security of PRF), and $\text{HYB}(\pi) \approx s_{\text{IDEAL}}(\pi)$ (programming can fail).
How the simulator works

- Let’s look again at the case of a **passively** corrupted server

 Define $\text{HYB}(\pi, i)$ to be the same as the real distribution but such that $R_p$ is random
How the simulator works

- Let’s look again at the case of a passively corrupted server

Define $\text{HYB}(\pi, \emptyset)$ to be the same as the real distribution but such that $R_p$ is random

- $\text{HYB}(\pi, \emptyset) \approx_c \text{REAL}(\pi, \emptyset)$ (by security of PRF)
How the simulator works

- Let’s look again at the case of a **passively** corrupted server

\[ \tilde{T}' \leftarrow \mathbb{Z}_M^\ell \]

Sample \( R'_p \) at random; for each \( i_j \) compute the block \( b \) and **program** \( \mathcal{H}(R'_p||b) := \sum \tilde{T}[\text{matching indexes in } b] \)

- Define \( \text{HYB}(\pi, \Omega) \) to be the same as the real distribution but such that \( R_p \) is random

\[
\begin{align*}
\text{HYB}(\pi, \Omega) &\approx_c \text{REAL}(\pi, \Omega) \quad \text{(by security of PRF)} \\
\text{HYB}(\pi, \Omega) &\approx_s \text{IDEAL}(\pi, \Omega) \quad \text{(programming can fail)}
\end{align*}
\]
Roadmap

- Security Definition
- Passive Security
- Proof Idea
- Construction
- Active Security

Rest of this Talk
Active security

- What about active corruption?

We need to verify correctness of \( \tilde{T} \)’s answers. To do so, we let the server outsource a succinct commitment to \( \tilde{T} \) and ask the server to open the values for the matching locations. Note that the server could still cheat by always declaring a "no match" (we avoid this via zero-knowledge sets).

We need to ensure that the server can search only \( p \)'s for which it has a trapdoor. For this, the \( \pi \) query must have active security.

We need to ensure that the text \( \tilde{T} \) computed by the server has associated a well-defined text \( T \). This requires expensive cut-and-choose techniques, which we avoid by a smart "on-the-fly" verification trick.
Active security

- What about active corruption?
- We need to verify correctness of cat’s answers
Active security

- What about active corruption?
- We need to verify correctness of \( \tilde{T} \)'s answers
  - To do so we let \( \tilde{T} \) outsource a succinct commitment to \( \tilde{T} \) and ask the server to open the values for the matching locations.
Active security

- What about active corruption?
- We need to verify correctness of \( T \)’s answers
  - To do so we let \( \hat{T} \) outsource a succinct commitment to \( \tilde{T} \) and ask the server to open the values for the matching locations
  - Note that \( \hat{T} \) could still cheat by always declaring a “no match” (we avoid this via zero-knowledge sets)
Active security

- What about active corruption?
- We need to verify correctness of \( \tilde{T} \)’s answers
  - To do so we let \( \text{Alice} \) outsource a succinct commitment to \( \tilde{T} \) and ask the server to open the values for the matching locations
  - Note that \( \text{Cyrus} \) could still cheat by always declaring a “no match” (we avoid this via zero-knowledge sets)
- We need to ensure \( \text{Alex} \) can search only \( p \)’s for which it has a trapdoor
Active security

- What about active corruption?
  - We need to verify correctness of $T$’s answers
    - To do so we let $T$ outsource a succinct commitment to $\tilde{T}$ and ask the server to open the values for the matching locations
    - Note that $T$ could still cheat by always declaring a “no match” (we avoid this via zero-knowledge sets)
  - We need to ensure $T$ can search only $p$’s for which it has a trapdoor
    - For this $\pi_{\text{Query}}$ must have active security
Active security

- What about active corruption?
- We need to verify correctness of the server’s answers.
  - To do so we let the server outsource a succinct commitment to $\tilde{T}$ and ask the server to open the values for the matching locations.
  - Note that the server could still cheat by always declaring a “no match” (we avoid this via zero-knowledge sets).
- We need to ensure that the server can search only $p$’s for which it has a trapdoor.
  - For this $\pi_{\text{Query}}$ must have active security.
- We need to ensure that the text $\tilde{T}$ computed by the server has associated a well defined text $T$.
Active security

- What about active corruption?

- We need to verify correctness of \( \tilde{T} \)’s answers
  
  - To do so we let \( \tilde{\mathcal{T}} \) outsource a succinct commitment to \( \tilde{T} \) and ask the server to open the values for the matching locations
  
  - Note that \( \tilde{\mathcal{T}} \) could still cheat by always declaring a “no match” (we avoid this via zero-knowledge sets)

- We need to ensure \( \tilde{\mathcal{T}} \) can search only \( p \)’s for which it has a trapdoor
  
  - For this \( \pi_{\text{Query}} \) must have active security

- We need to ensure that the text \( \tilde{T} \) computed by \( \tilde{\mathcal{T}} \) has associated a well defined text \( T \)
  
  - This requires expensive cut-and-choose techniques, which we avoid by a smart “on-the-fly” verification trick
Take-home message

- We give a simulation-based security definition for outsourced pattern matching
Take-home message

- We give a simulation-based security definition for outsourced pattern matching
- We construct a protocol with passive security (in the RO model) and sublinear communication complexity in the query phase (which is optimal)

Open problems for future work:
- An efficient construction in the standard model
- Extensions (pattern matching with wildcards, approximate pattern matching, hiding the length of the text/pattern)
Take-home message

- We give a simulation-based security definition for outsourced pattern matching.
- We construct a protocol with passive security (in the RO model) and sublinear communication complexity in the query phase (which is optimal).
- We explain how to modify the basic protocol to tolerate active adversaries.
Take-home message

- We give a simulation-based security definition for outsourced pattern matching
- We construct a protocol with passive security (in the RO model) and sublinear communication complexity in the query phase (which is optimal)
- We explain how to modify the basic protocol to tolerate active adversaries
- Open problems for future work:
Take-home message

- We give a simulation-based security definition for outsourced pattern matching
- We construct a protocol with passive security (in the RO model) and sublinear communication complexity in the query phase (which is optimal)
- We explain how to modify the basic protocol to tolerate active adversaries
- Open problems for future work:
  - An efficient construction in the standard model
Take-home message

- We give a simulation-based security definition for outsourced pattern matching
- We construct a protocol with passive security (in the RO model) and sublinear communication complexity in the query phase (which is optimal)
- We explain how to modify the basic protocol to tolerate active adversaries
- Open problems for future work:
  - An efficient construction in the standard model
  - Extensions (pattern matching with wildcards, approximate pattern matching, hiding the length of the text/pattern)
Thank you!