PASTA: Password-based Threshold Authentication

Peihan Miao

June 24, 2019

Joint work with Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee.
What problem is PASTA trying to solve?
OAuth
OAuth
OAuth
OAuth
OAuth

(Username, Password)
OAuth

Request

(username, password)

Google

Dropbox
OAuth

(request)

Username, Password

Digital Signature

Google

Token

Dropbox
JSON Web Token (JWT)
JSON Web Token (JWT)

Request

Digital Signature
or MAC

(Username, Password)

Google

Gmail
JSON Web Token (JWT)

Request

Digital Signature
or MAC

Google

Gmail
Kerberos
Kerberos
Kerberos

![Diagram of Kerberos process]

- Request
- (MAC)
- (Username, Password)

![Printer]

Process:
1. User requests access to a resource.
2. Server verifies the credentials provided (Username, Password).
3. Server generates a MAC for verified credentials.
4. Client receives the MAC to verify with their request.
5. Printer or other resource is validated by the server.
Kerberos

Request

(MAC)

(Username, Password)

Printer
Password-Based Token Generation
Password-Based Token Generation
Password-Based Token Generation

(Username, Password)
Password-Based Token Generation

Request

(Notification, Password)
Password-Based Token Generation

(Username, Password)
Password-Based Token Generation
Password Safety

(Username, Password)
Password Safety

(username, password)
Password Safety

Request
Username, Hash(pwd)

(Username, Password)
Password Safety

Request

Username, Hash(pwd)

Username, Hash(pwd)

(Username, Password)
Password Safety

[Diagram showing the process of requesting a username and hash of the password from a server, with a note for security measures.]

(Username, Password)
Password Safety

Request
Username, Hash(pwd)

(Username, Password)

Hash(Birthday)
Hash(PhoneNumber)
Hash(HomeAddress)
Password Safety

Request
Username, Hash(pwd)

(Username, Password)

Offline Dictionary Attack!
Hash(Birthday)
Hash(Phone Number)
Hash(Home Address)
Password Safety

(Username, Password)
Password Safety

(request)

(Username, Password)
Password Safety

(request)

(Username, Password)
Password Safety

(Username, Password)
Password Safety
Password Safety
Password Safety
Password Safety

(Username, Birthday)
(Username, PhoneNumber)
(Username, HomeAddress)

::
Password Safety

Request

(Username, Birthday)
(Username, PhoneNumber)
(Username, HomeAddress)
Password Safety

Request

(Username, Birthday)
(Username, PhoneNumber)
(Username, HomeAddress)

?
Password Safety

Request

(Username, Birthday)
(Username, PhoneNumber)
(Username, HomeAddress)

Offline Dictionary Attack!
Our Goal
Our Goal
Our Goal
Our Goal
Our Goal
Our Goal

- If $< t$ servers are corrupted:
Our Goal

- If \(< t\) servers are corrupted:
  - Secret key is secure
Our Goal

- If $< t$ servers are corrupted:
  - Secret key is secure
  - Password is secure (against offline attack)
Our Contribution
Our Contribution

• Password-based Threshold Authentication (PbTA)
  • Motivate the problem
  • Formalize a game-based security definition
Our Contribution

• Password-based Threshold Authentication (PbTA)
  • Motivate the problem
  • Formalize a game-based security definition

• PASTA: a framework of construction for various types of tokens
  • Minimal round complexity
  • Low computational & communication complexity
Our Contribution

- Password-based Threshold Authentication (PbTA)
  - Motivate the problem
  - Formalize a game-based security definition

- PASTA: a framework of construction for various types of tokens
  - Minimal round complexity
  - Low computational & communication complexity

- Implementations and experiments
Our Contribution

- **Password-based Threshold Authentication (PbTA)**
  - Motivate the problem
  - Formalize a game-based security definition

- **PASTA**: a framework of construction for various types of tokens
  - Minimal round complexity
  - Low computational & communication complexity

- Implementations and experiments
Global Setup
Global Setup
Global Setup
Global Setup

Any $t$ servers can recover the secret key.

(In this example, $t = 3$.)
Global Setup

Any $t$ servers can recover the secret key.

(In this example, $t = 3$.)
Global Setup

Any $t$ servers can recover the secret key.

Any $t - 1$ servers cannot recover the secret key.

(In this example, $t = 3$.)
User Sign Up
User Sign Up

(Username, Password)
Request for Token
Request for Token

(Username, Password)
Request for Token

(Username, Password)
Request for Token

(Username, Password)

Any $t$ valid responses can recover the token.

(In this example, $t = 3$.)
Request for Token

(Username, Password)

Any $t$ valid responses can recover the token.

(In this example, $t = 3$.)
Request for Token

Any $t$ valid responses can recover the token.

Any $t - 1$ servers cannot recover the password (by offline attack) or forge a token.

(In this example, $t = 3$.)
Overview

• What problem is PASTA trying to solve?

• How does PASTA work?

• Is it practical?
Our Approach
Our Approach

• Non-Interactive Threshold Token Generation (NITTG)
  • Symmetric key based MAC [NPR’99, MPSWW’02]
  • Public key (DDH) based MAC [NPR’99]
  • RSA based digital signature [Shoup’00]
  • Pairing based digital signature [BLS’01, Boldyreva’03]

• Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX’17]
  • **Formalize** game-based definition for our needs
  • **Prove** security for the construction
Our Approach

- Non-Interactive Threshold Token Generation (NITTG)
  - Symmetric key based MAC [NPR’99, MPSWW’02]
  - Public key (DDH) based MAC [NPR’99]
  - RSA based digital signature [Shoup’00]
  - Pairing based digital signature [BLS’01, Boldyreva’03]

- Threshold Oblivious Pseudorandom Function (TOPRF) [KKX’17]
  - Formalize game-based definition for our needs
  - Prove security for the construction

- Generic construction of PASTA from NITTG + TOPRF
Non-Interactive Threshold Token Generation (NITTG)
Non-Interactive Threshold Token Generation (NITTG)
Non-Interactive Threshold Token Generation (NITTG)
Non-Interactive Threshold Token Generation (NITTG)
Non-Interactive Threshold Token Generation (NITTG)
Non-Interactive Threshold Token Generation (NITTG)

Any $t$ "partial" tokens can recover the token.

(In this example, $t = 3$.)
Non-Interactive Threshold Token Generation (NITTG)

Any $t$ “partial” tokens can recover the token.

(In this example, $t = 3$.)
Non-Interactive Threshold Token Generation (NITTG)

Any $t$ “partial” tokens can recover the token.

Any $t - 1$ key shares cannot recover the key or forge a token.

(In this example, $t = 3$.)
Non-Interactive Threshold Token Generation (NITTG)

Any $t$ “partial” tokens can recover the token.

Any $t - 1$ key shares cannot recover the key or forge a token.

(In this example, $t = 3$.)
Naïve Solution

(Username, Password)
Naïve Solution

(Username, Password)
Naïve Solution

(Username, Password)
Naïve Solution

(request (Username, Hash(pwd))

(request (Username, Hash(pwd)))

(request (Username, Hash(pwd)))

(request (Username, Hash(pwd)))

(request (Username, Hash(pwd)))

(request (Username, Hash(pwd)))

(request (Username, Hash(pwd)))
Naïve Solution
Naïve Solution

Secret key is secure!
Naïve Solution

Secret key is secure!

Password is not secure (against offline dictionary attack)!
Naïve Solution

Secret key is secure!

Password is not secure (against offline dictionary attack)!
Naïve Solution

Secret key is secure!

Password is not secure (against offline dictionary attack)!
Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX’17]
Threshold Oblivious Pseudorandom Function (TOPRF) [JKXX'17]
Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX’17]
Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX’17]

Input: \( x \)

\[ y = F_k(x) \]

Any \( t \) valid responses can recover \( y \).
Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX’17]

Input: \( x \)
\[
y = F_k(x)
\]

Any \( t \) valid responses can recover \( y \).

\( k \) is hidden to the client.
\( x \) is hidden to the servers.
Any \( t - 1 \) key shares cannot recover the key or compute \( F_k(x) \).
Our Approach

• Non-Interactive Threshold Token Generation (NITTG)
  • Symmetric key based MAC [NPR’99, MPSWW’02]
  • Public key (DDH) based MAC [NPR’99]
  • RSA based digital signature [Shoup’00]
  • Pairing based digital signature [BLS’01, Boldyreva’03]

• Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX’17]
  • Formalize game-based definition for our needs
  • Prove security for the construction

• Generic construction of PASTA from NITTG + TOPRF
Global Setup
Global Setup
Global Setup

- 1 shared key for NITTG
- 1 shared key for TOPRF
User Sign Up
User Sign Up

(Username, Password)
User Sign Up

Input: $x = (\text{Username}, \text{Password})$

$y$
User Sign Up

Input: $x = (\text{Username, Password})$
User Sign Up

Input: $x = (\text{Username}, \text{Password})$

$y$

Sign Up

$y$

Sign Up

$y$

Sign Up

$y$

Sign Up

$y$

Username, $y$

Username, $y$

Username, $y$

Username, $y$

Username, $y$

Username, $y$
Request for Token
Request for Token

(Username, Password)
Request for Token

Input: $x = (\text{Username, Password})$
Request for Token – Cont’d

Input: $x = (\text{Username, Password})$

$y$
Request for Token – Cont’d

Input: $x = (\text{Username, Password})$

Diagram shows a flow of information from a laptop to a series of database entries labeled with Username and y.
Request for Token – Cont’d

Input: $x = \text{(Username, Password)}$

Diagram:
- A laptop labeled with $x = \text{(Username, Password)}$
- Multiple output nodes labeled with $\text{Username, y}$
Request for Token – Cont’d

Input: $x = (\text{Username, Password})$

Any $t - 1$ servers cannot compute $F_k(x)$. 
Request for Token – Cont’d

Input: $x = (\text{Username, Password})$

Any $t - 1$ servers cannot compute $F_k(x)$.

How to reduce round complexity?
Request for Token – Minimal Interaction
Request for Token – Minimal Interaction

(Username, Password)
Request for Token – Minimal Interaction

Input: $x = (\text{Username}, \text{Password})$
Request for Token – Minimal Interaction

Input: \( x = (\text{Username}, \text{Password}) \)
Request for Token – Minimal Interaction

Input: \( x = (\text{Username}, \text{Password}) \)

Any \( t \) valid responses can recover the token.
Request for Token – Minimal Interaction

Input: \( x = (\text{Username, Password}) \)

Any \( t \) valid responses can recover the token.
Request for Token – Minimal Interaction

Input: $x = (\text{Username, Password})$

Any $t$ valid responses can recover the token.
Any $t - 1$ servers cannot recover the password (by offline attack) or forge a token.
Request for Token
Request for Token

(Username, Guess)
Request for Token

Input: $x = (\text{Username, Guess})$
Request for Token

Input: $x = (\text{Username}, \text{Guess})$

$y'$

$\text{Enc}_y(\text{Username})$

$\text{Enc}_y(\text{Guess})$

$\text{Enc}_y(y)$
Request for Token

Input: $x = (\text{Username, Guess})$

$y'$
User Sign Up – A Fix
User Sign Up – A Fix

(Username, Password)
User Sign Up – A Fix

Input: $x = (\text{Username}, \text{Password})$
User Sign Up – A Fix

Input: $x = (\text{Username, Password})$

$y$
User Sign Up – A Fix

Input: \( x = (\text{Username, Password}) \)

\[ y \]

\[ k_1, k_2, k_3, k_4, k_5 \]
User Sign Up – A Fix

Input: $x = (\text{Username, Password})$

$y \leftarrow k_1, k_2, k_3, k_4, k_5$
User Sign Up – A Fix

Input: $x = (\text{Username}, \text{Password})$

$y \downarrow$

$k_1, k_2, k_3, k_4, k_5$

Sign Up

Username, $k_1$

Username, $k_2$

Username, $k_3$

Username, $k_4$

Username, $k_5$
Request for Token – A Fix

(Username, Password)
Request for Token – A Fix

Input: $x = (\text{Username, Password})$
Request for Token – A Fix

Input: $x = (\text{Username, Password})$

$y$

$k_1, k_2, k_3, k_4, k_5$
Request for Token – A Fix

Input: \( x = (\text{Username, Password}) \)

\[ y \]

\[ k_1, k_2, k_3, k_4, k_5 \]
Request for Token – A Fix

Input: $x = (\text{Username, Password})$

$y$

$k_1, k_2, k_3, k_4, k_5$

Any $t$ valid responses can recover the token.
Request for Token – A Fix

Input: $x = (\text{Username, Password})$

$y$

$k_1, k_2, k_3, k_4, k_5$

Any $t$ valid responses can recover the token.

Any $t - 1$ servers cannot recover the password (by offline attack) or forge a token.
Request for Token – A Fix

Input: $x = (\text{Username, Password})$

$y$

$k_1, k_2, k_3, k_4, k_5$

Any $t$ valid responses can recover the token.

Any $t - 1$ servers cannot recover the password (by offline attack) or forge a token?
User Sign Up

Input: \( x = (\text{Username, Password}) \)

\[ y \]

\( k_1, k_2, k_3, k_4, k_5 \)
User Sign Up

Input: $x = \text{(Username, Password)}$

Output: $y = k_1, k_2, k_3, k_4, k_5$

How to reduce round complexity?
User Sign Up – Minimal Interaction
User Sign Up – Minimal Interaction
User Sign Up – Minimal Interaction

Input: $x = (\text{Username}, \text{Password})$
User Sign Up – Minimal Interaction

Input: $x = (\text{Username}, \text{Password})$

$y$

$k_1, k_2, k_3, k_4, k_5$
User Sign Up – Minimal Interaction

Input: \( x = (\text{Username, Password}) \)

Output: \( y = (k_1, k_2, k_3, k_4, k_5) \)
User Sign Up – Minimal Interaction

Input: $x = \text{(Username, Password)}$

Output: $y = k_1, k_2, k_3, k_4, k_5$

- Username, $k_1$, TOPRF key
- Username, $k_2$, TOPRF key
- Username, $k_3$, TOPRF key
- Username, $k_4$, TOPRF key
- Username, $k_5$, TOPRF key
Overview

• What problem is PASTA trying to solve?

• How does PASTA work?

• Is it practical?
Implementation
Implementation

- Non-Interactive Threshold Token Generation (NITTG) for
  - Symmetric key based MAC
  - Public key (DDH) based MAC
  - RSA based digital signature *(first implementation)*
  - Pairing based digital signature *(first implementation)*
Implementation

• Non-Interactive Threshold Token Generation (NITTTG) for
  • Symmetric key based MAC
  • Public key (DDH) based MAC
  • RSA based digital signature (first implementation)
  • Pairing based digital signature (first implementation)

• Threshold Oblivious Pseudorandom Function (TOPRF)
Implementation

- Non-Interactive Threshold Token Generation (NITTG) for
  - Symmetric key based MAC
  - Public key (DDH) based MAC
  - RSA based digital signature (first implementation)
  - Pairing based digital signature (first implementation)

- Threshold Oblivious Pseudorandom Function (TOPRF)

- PASsword-based Threshold Authentication (PASTA)
## Experiment

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**LAN network:** 10Gbps, 0.1ms RTT latency
## Experiment

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**LAN network:** 10Gbps, 0.1ms RTT latency
Plain Setting

(Username, Password)

Username, Hash(pwd)
Plain Setting

Request
Username, Hash(pwd)

Username, Hash(pwd)
Plain Setting

(Request)

Username, Hash(pwd)

(Username, Password)

Username, Hash(pwd)
Plain Setting

(Username, Password)
Plain Setting

Request
Username, Hash(pwd)

(Username, Password)

User_pwd)
Plain Setting

What’s the extra cost of protecting secret key and password?
## Experiment

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**LAN network**: 10Gbps, 0.1ms RTT latency
## Experiment

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**LAN network:** 10Gbps, 0.1ms RTT latency
## Experiment

<table>
<thead>
<tr>
<th>$(n, t)$</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**LAN network**: 10Gbps, 0.1ms RTT latency
We showed public key operations are theoretically necessary to achieve our goal.

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

LAN network: 10Gbps, 0.1ms RTT latency
## Experiment

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>80.2</td>
<td>81.4</td>
<td>81.4</td>
<td>81.7</td>
<td>81.4</td>
<td>81.7</td>
<td>82.7</td>
<td>81.4</td>
<td>82.4</td>
<td>84.2</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>80.4</td>
<td>81.8</td>
<td>81.8</td>
<td>82.2</td>
<td>81.8</td>
<td>82.2</td>
<td>83.6</td>
<td>81.9</td>
<td>83.1</td>
<td>85.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>80.4</td>
<td>81.8</td>
<td>81.8</td>
<td>82.2</td>
<td>81.8</td>
<td>82.2</td>
<td>83.6</td>
<td>81.9</td>
<td>83.1</td>
<td>85.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>91.3</td>
<td>94.6</td>
<td>94.6</td>
<td>95.0</td>
<td>94.6</td>
<td>95.3</td>
<td>98.6</td>
<td>94.5</td>
<td>96.9</td>
<td>102.8</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**WAN network**: 40Mbps, 80ms RTT latency
## Experiment

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>80.2</td>
<td>81.4</td>
<td>81.4</td>
<td>81.7</td>
<td>81.4</td>
<td>81.7</td>
<td>82.7</td>
<td>81.4</td>
<td>82.4</td>
<td>84.2</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>80.4</td>
<td>81.8</td>
<td>81.8</td>
<td>82.2</td>
<td>81.8</td>
<td>82.2</td>
<td>83.6</td>
<td>81.9</td>
<td>83.1</td>
<td>85.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>80.4</td>
<td>81.8</td>
<td>81.8</td>
<td>82.2</td>
<td>81.8</td>
<td>82.2</td>
<td>83.6</td>
<td>81.9</td>
<td>83.1</td>
<td>85.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>91.3</td>
<td>94.6</td>
<td>94.6</td>
<td>95.0</td>
<td>94.6</td>
<td>95.3</td>
<td>98.6</td>
<td>94.5</td>
<td>96.9</td>
<td>102.8</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**WAN network**: 40Mbps, 80ms RTT latency
<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(n, t)</th>
<th>plain</th>
<th>(2,2)</th>
<th>(3,2)</th>
<th>(3,3)</th>
<th>(6,2)</th>
<th>(6,3)</th>
<th>(6,6)</th>
<th>(10,2)</th>
<th>(10,5)</th>
<th>(10,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>80.2</td>
<td>81.4</td>
<td>81.4</td>
<td>81.7</td>
<td>81.4</td>
<td>81.7</td>
<td>82.7</td>
<td>81.4</td>
<td>82.4</td>
<td>84.2</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>80.4</td>
<td>81.8</td>
<td>81.8</td>
<td>82.2</td>
<td>81.8</td>
<td>82.2</td>
<td>83.6</td>
<td>81.9</td>
<td>83.1</td>
<td>85.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>80.4</td>
<td>81.8</td>
<td>81.8</td>
<td>82.2</td>
<td>81.8</td>
<td>82.2</td>
<td>83.6</td>
<td>81.9</td>
<td>83.1</td>
<td>85.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>91.3</td>
<td>94.6</td>
<td>94.6</td>
<td>95.0</td>
<td>94.6</td>
<td>95.3</td>
<td>98.6</td>
<td>94.5</td>
<td>96.9</td>
<td>102.8</td>
</tr>
<tr>
<td>(n, t)</td>
<td>plain</td>
<td>(2,2)</td>
<td>(3,2)</td>
<td>(3,3)</td>
<td>(6,2)</td>
<td>(6,3)</td>
<td>(6,6)</td>
<td>(10,2)</td>
<td>(10,5)</td>
<td>(10,10)</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.6</td>
<td>1.3</td>
<td>1.6</td>
<td>2.7</td>
<td>1.3</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>1.7</td>
<td>2.1</td>
<td>1.7</td>
<td>2.1</td>
<td>3.5</td>
<td>1.7</td>
<td>3.0</td>
<td>5.4</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>14.5</td>
<td>15.0</td>
<td>14.6</td>
<td>15.1</td>
<td>18.6</td>
<td>14.5</td>
<td>16.8</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Our round complexity is optimal!
# Time Breakdown

<table>
<thead>
<tr>
<th></th>
<th>plain</th>
<th>(10,2)</th>
<th>Server</th>
<th>Client</th>
<th>(10,6)</th>
<th>Server</th>
<th>Client</th>
<th>(10,10)</th>
<th>Server</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sym-MAC</td>
<td>0.1</td>
<td>1.3</td>
<td>0.2</td>
<td>1.0</td>
<td>2.7</td>
<td>0.2</td>
<td>2.3</td>
<td>4.1</td>
<td>0.2</td>
<td>3.7</td>
</tr>
<tr>
<td>Public-MAC</td>
<td>0.4</td>
<td>1.7</td>
<td>0.4</td>
<td>1.2</td>
<td>3.5</td>
<td>0.4</td>
<td>2.9</td>
<td>5.4</td>
<td>0.4</td>
<td>4.6</td>
</tr>
<tr>
<td>Pairing-Sig</td>
<td>0.4</td>
<td>1.7</td>
<td>0.4</td>
<td>1.2</td>
<td>3.5</td>
<td>0.4</td>
<td>2.9</td>
<td>5.4</td>
<td>0.4</td>
<td>4.6</td>
</tr>
<tr>
<td>RSA-Sig</td>
<td>11.3</td>
<td>14.5</td>
<td>11.4</td>
<td>3.0</td>
<td>18.5</td>
<td>11.5</td>
<td>6.8</td>
<td>22.6</td>
<td>11.5</td>
<td>10.7</td>
</tr>
</tbody>
</table>

Token request performance (ms) for an average of 10,000 token requests

**LAN network:** 10Gbps, 0.1ms RTT latency
Summary

- What problem is PASTA trying to solve?

- How does PASTA work?

- Is it practical?
Summary

• What problem is PASTA trying to solve?
  • Password-based threshold authentication
  • Protect both secret key and passwords

• How does PASTA work?

• Is it practical?
Summary

• What problem is PASTA trying to solve?
  • Password-based threshold authentication
  • Protect both secret key and passwords

• How does PASTA work?
  • Generic construction from NITTG + TOPRF
  • Framework for various types of tokens

• Is it practical?
Request for Token – A Fix

Input: \( x = (\text{Username, Password}) \)

\[ y \]

\( k_1, k_2, k_3, k_4, k_5 \)

Any \( t \) valid responses can recover the token.

Any \( t - 1 \) servers cannot recover the password (by offline attack) or forge a token?