Watermarking PRF against Quantum Adversaries

Fuyuki Kitagawa (NTT Corporation)
Ryo Nishimaki (NTT Corporation)
Software watermarking
Software watermarking

We can embed a mark into a program
Software watermarking

We can embed a mark into a program. If the mark is removed, the program is destroyed.
Software watermarking

We can embed a mark into a program.

If the mark is removed, the program is destroyed.

- The purpose: proving ownership, preventing illegal copies, and so on.
Watermarking crypto programs

- Software watermarking can deal with only unlearnable programs
  - If a program is learnable, the mark is removed by learning original program
Watermarking crypto programs

• Software watermarking can deal with only unlearnable programs
  - If a program is learnable, the mark is removed by learning original program

• Previous works have focused on watermarking crypto programs
  - Especially, most of them studied watermarking PRF
    - Simplest, but sufficient for many other crypto primitives
Watermarking crypto programs

- Software watermarking can deal with only unlearnable programs
  - If a program is learnable, the mark is removed by learning original program
- Previous works have focused on watermarking crypto programs
  - Especially, most of them studied watermarking PRF
    - Simplest, but sufficient for many other crypto primitives
- Application to quantum crypto [KNY21,ALLZZ21]
  - By combining with quantum money, we can construct secure software leasing
This work

• Watermarking PRF against quantum adversaries
Our results

1. We define watermarking PRF against quantum adversaries

2. We construct watermarking PRF against quantum adversaries
Our results

1. We define watermarking PRF against quantum adversaries
   • We define unremovability against adversaries who output quantum program

2. We construct watermarking PRF against quantum adversaries
Our results

1. We define watermarking PRF against quantum adversaries
   - We define unremovability against adversaries who output quantum program

2. We construct watermarking PRF against quantum adversaries
   - Secret extractable scheme based on LWE
   - Public extractable scheme based on IO
Our results

1. We define watermarking PRF against quantum adversaries
   - We define unremovability against adversaries who output quantum program

2. We construct watermarking PRF against quantum adversaries
   - Secret extractable scheme based on LWE
   - Public extractable scheme based on IO
     - Our construction methodology is highly general and can be extended to watermarking other primitives such as PKE
Technical overview

- The biggest issue is that quantum programs are stateful programs
  - It was pointed out by Zhandry [Zha20] in the context of traitor tracing
  - Classical traitor tracing/watermarking assumes pirate programs are stateless
    - It is reasonable since we can rewind pirate programs
  - It is impossible to rewind quantum program
Technical overview

• The biggest issue is that quantum programs are stateful programs
  - It was pointed out by Zhandry [Zha20] in the context of traitor tracing
  - Classical traitor tracing/watermarking assumes pirate programs are stateless
    - It is reasonable since we can rewind pirate programs
  - It is impossible to rewind quantum program

• We need to define unremovability taking it into consideration

• We need to construct our scheme taking it into consideration
Technical overview

• The biggest issue is that quantum programs are stateful programs
  - It was pointed out by Zhandry [Zha20] in the context of traitor tracing
  - Classical traitor tracing/watermarking assumes pirate programs are stateless
    - It is reasonable since we can rewind pirate programs
  - It is impossible to rewind quantum program

• We need to define unremovability taking it into consideration
  - We can use Zhandry [Zha20]’s technique

• We need to construct our scheme taking it into consideration
Technical overview

• The biggest issue is that quantum programs are stateful programs
  - It was pointed out by Zhandry [Zha20] in the context of traitor tracing
  - Classical traitor tracing/watermarking assumes pirate programs are stateless
    - It is reasonable since we can rewind pirate programs
    - It is impossible to rewind quantum program

• We need to define unremovability taking it into consideration
  - We can use Zhandry [Zha20]’s technique

• We need to construct our scheme taking it into consideration
  - We cannot use Zhandry [Zha20]’s technique
  - We propose new extraction method
Syntax and functionality preserving

• Essentially the same as (single-key) traceable PRF [GKWW21]
• Consists of (Gen, Eval, Mark, Ext)
Syntax and functionality preserving

- Essentially the same as (single-key) traceable PRF [GKWW21]
- Consists of (Gen, Eval, Mark, Ext)

\[ 1^\lambda \xrightarrow{\text{Gen}} (\text{prfk}, x_k) \]

\[ (\text{prfk}, x) \xrightarrow{\text{Eval}} y \]
Syntax and functionality preserving

- Essentially the same as (single-key) traceable PRF [GKWW21]
- Consists of (Gen,Eval,Mark,Ext)

\[ 1^\lambda \xrightarrow{} \text{Gen} \xrightarrow{} (\text{prfk},x_k) \quad \text{prfk,}m \xrightarrow{} \text{Mark} \xrightarrow{} C' \]

\[(\text{prfk},x) \xrightarrow{} \text{Eval} \xrightarrow{} y \]
Syntax and functionality preserving

- Essentially the same as (single-key) traceable PRF [GKWW21]
- Consists of (Gen, Eval, Mark, Ext)

1^\lambda \xrightarrow{} \text{Gen} \rightarrow (\text{prfk}, x_k) \quad \text{prfk}, m \xrightarrow{} \text{Mark} \rightarrow C'

(pfk,x) \xrightarrow{} \text{Eval} \rightarrow y \quad x_k, |\psi\rangle \xrightarrow{} \text{Ext} \rightarrow m
Syntax and functionality preserving

- Essentially the same as (single-key) traceable PRF \([\text{GKWW21}]\)
- Consists of (Gen, Eval, Mark, Ext)

\[
\begin{align*}
1^\lambda & \rightarrow \text{Gen} \rightarrow (\text{prfk}, xk) \quad \text{prfk}, m & \rightarrow \text{Mark} \rightarrow C' \\
(\text{prfk}, x) & \rightarrow \text{Eval} \rightarrow y \quad xk, |\psi\rangle & \rightarrow \text{Ext} \rightarrow m
\end{align*}
\]

- Functionality preserving:

\[C'(x) = \text{Eval}(\text{prfk}, x) \text{ for almost all inputs } x\]
Unremovability

- The definition is roughly as follows

\[ C' = \text{Mark}(\text{prfk}, m) \]
Unremovability

• The definition is roughly as follows

\[ C' = \text{Mark}(\text{prfk}, m) \]

No adversary can generate \( |\psi\rangle \) s.t.

- \( |\psi\rangle \) is “good” in the sense that its functionality is close to \( \text{Eval}(\text{prfk}, \cdot) \)
- \( \text{Ext} \) fails to extract \( m \)
Unremovability

- The definition is roughly as follows

\[
C' = \text{Mark}(\text{prfk}, m)
\]

No adversary can generate \(|\psi\rangle\) s.t.

- \(|\psi\rangle\) is “good” in the sense that its functionality is close to \(\text{Eval}(\text{prfk}, \cdot)\)
- \(\text{Ext}\) fails to extract \(m\)

- To make the definition rigorous, we have to define “good” more concretely
How to define “good” quantum program?

- Define as quantum programs breaking weak PRF security [GKWW21]
How to define “good” quantum program?

- Define as quantum programs breaking weak PRF security [GKWW21]
- We have to take the stateful nature of them into consideration
How to define “good” quantum program?

• Define as quantum programs breaking weak PRF security [GKWW21]

• We have to take the stateful nature of them into consideration
  - We have to measure the advantage for breaking weak PRF security
How to define “good” quantum program?

- Define as quantum programs breaking weak PRF security [GKWW21]
- We have to take the stateful nature of them into consideration
  - We have to measure the advantage for breaking weak PRF security
  - The measurement itself might destroy the quantum program
How to define “good” quantum program?

- Define as quantum programs breaking weak PRF security [GKWW21]

- We have to take the stateful nature of them into consideration
  - We have to measure the advantage for breaking weak PRF security
  - The measurement itself might destroy the quantum program
  - If we once confirm a quantum program is a good program, the post measurement state might not be a good program
How to define “good” quantum program?

- Define as quantum programs breaking weak PRF security [GKWW21]
- We have to take the stateful nature of them into consideration
  - We have to measure the advantage for breaking weak PRF security
  - The measurement itself might destroy the quantum program
  - If we once confirm a quantum program is a good program, the post measurement state might not be a good program
- We use the notion of Live by Zhandry[Zha20] as the notion of “good”
How to define “good” quantum program?

- Define as quantum programs breaking weak PRF security [GKWW21]

- We have to take the stateful nature of them into consideration
  - We have to measure the advantage for breaking weak PRF security
  - The measurement itself might destroy the quantum program
  - If we once confirm a quantum program is a good program, the post measurement state might not be a good program

- We use the notion of Live by Zhandry [Zha20] as the notion of “good”
  - It is defined by using projective implementation (ProjImp)
In the context of classical traitor tracing or watermarking, we can assume that a program is not really meaningful. The biggest issue is that it does not consider the stateful nature of quantum programs. In this work, we allow to take the public tag \( \tau \) as stated above. Let \( D_{\text{wprf}} \) be a distribution of \( \langle b,x,y \rangle \) where \( b \in \{0,1\}, x \in \text{Dom}, y \in \text{Ran}, y_1 \leftarrow \text{Eval}(\text{prfk},x) \) is output. On the other hand, we consider both the public extraction and the private extraction settings. Thus, the extraction key \( \text{pp} \) is given and an extraction key \( \text{pk} \) is kept. Then, for a quantum state \( \langle \psi \rangle \), the overall extraction algorithm Extract is unavoidable. For a more detailed discussion on the syntax, see the discussion in Section 3.1.
On quantum programs

- Success probability for breaking weak PRF is defined as follows

\[ D_{\text{wprf}}: b \leftarrow \{0,1\}, x \leftarrow \text{Dom}, y_0 \leftarrow \text{Ren}, y_1 \leftarrow \text{Eval(prfk},x), \text{output } (b,x,y_b) \]

Success probability is the probability that

\[ (b,x,y_b) \leftarrow D_{\text{wprf}} \]

\[ (x,y_b) \rightarrow \psi \rightarrow b \]

- \(|\psi\rangle\) can be seen as super-position of programs with different success probabilities w.r.t \(D_{\text{wprf}}\)

\[ |\psi\rangle = \sum_p \alpha_p |\psi_p\rangle, \text{ where } |\psi_p\rangle \text{ has success probability } p \text{ w.r.t } D_{\text{wprf}} \text{ and } \sum_p \alpha_p^2 = 1 \]
ProjImp and live quantum programs [Zha20]

- Quantum program can be seen as a super-position of many programs
- ProjImp measures the success probability of one of them
- Concretely, it is defined as follows

$$|\psi\rangle = \sum_p \alpha_p |\psi_p\rangle$$
ProjImp and live quantum programs [Zha20]

- Quantum program can be seen as a super-position of many programs
- ProjImp measures the success probability of one of them
- Concretely, it is defined as follows

\[ |\psi\rangle = \sum_p \alpha_p |\psi_p\rangle \]

Outcome \( p \) with probability \( \alpha_p^2 \)

ProjImp(\( D_{wprf} \))
ProjImp and live quantum programs [Zha20]

- Quantum program can be seen as a super-position of many programs
- ProjImp measures the success probability of one of them
- Concretely, it is defined as follows

\[ |\psi_p\rangle = \sum_p \alpha_p |\psi_p\rangle \]

Success probability \( p \)

Outcome \( p \)

with probability \( \alpha_p^2 \)

ProjImp(\( D_{\text{wprf}} \))
ProjImp and live quantum programs [Zha20]

- Quantum program can be seen as a super-position of many programs
- ProjImp measures the success probability of one of them
- Concretely, it is defined as follows

\[ |\psi\rangle = \sum_p \alpha_p |\psi_p\rangle \]

- \( |\psi\rangle \) is **Live** if the result \( p \) of ProjImp\((D_{wprf})\) is significantly greater than 1/2
  - For classical programs, it is the same as classical “good” [GKWW21]
Unremovability

No adversary can generate $|\psi\rangle$ s.t.
- $|\psi\rangle$ is a Live quantum program
- Ext fails to extract $m$

$C' = \text{Mark(prfk,m)}$
How to extract mark?

- Our goal is

\[ C' = \text{Mark}(\text{prfk}, m) \]
How to extract mark?

- Our goal is

\[ C' = \text{Mark}(\text{prfk}, m) \]

**Diagram:**
- **Input:** `prfk m`
- **Output:** `m`
- **Process:**
  - `Mark`: Function to extract mark from a quantum circuit.
  - `Ext`: Extraction algorithm.
  - `Live`: Circuit outputs a message.
  - Apply several tests on success probability to verify the extracted message.
How to extract mark?

• Our goal is

$$C' = \text{Mark} (\text{prfk}, \text{m})$$

• The set of applicable tests is highly limited compared to classical case
  - Due to the stateful nature, a test can destroy the quantum program
Difficulty

• In watermarking, an embedded mark is chosen from super-poly size set
Difficulty

- In watermarking, an embedded mark is chosen from super-poly size set.
  - Extraction of mark is bit-by-bit manner.
**Difficulty**

- In watermarking, an embedded mark is chosen from super-poly size set.
  - Extraction of mark is bit-by-bit manner.
- We need to realize a test $T_i$ for every $i$ s.t.
  - $T_i$ can be used to extract the $i$-th bit of the mark.
  - $T_i$ does not destroy the quantum program.
Difficulty

- In watermarking, an embedded mark is chosen from super-poly size set
  - Extraction of mark is bit-by-bit manner
- We need to realize a test $T_i$ for every $i$ s.t.
  - $T_i$ can be used to extract the $i$-th bit of the mark
  - $T_i$ does not destroy the quantum program
Our idea

- We use reverse projective property of ProjImp
- Let $D_{\text{fail}}$ be the distribution generates $(b,x,y) \leftarrow D_{\text{wprf}}$ and outputs $(1-b,x,y)$
  - $\text{ProjImp}(D_{\text{fail}})$ measures failure probability
Our idea

- We use reverse projective property of ProjImp
- Let $D_{\text{fail}}$ be the distribution generates $(b,x,y) \leftarrow D_{\text{wprf}}$ and outputs $(1-b,x,y)$
  - ProjImp($D_{\text{fail}}$) measures failure probability

$$|\psi\rangle = \sum_p \alpha_p |\psi_p\rangle$$
Our idea

- We use reverse projective property of ProjImp
- Let $D_{\text{fail}}$ be the distribution generates $(b,x,y) \xleftarrow{} D_{\text{wprf}}$ and outputs $(1-b,x,y)$.
  - ProjImp($D_{\text{fail}}$) measures failure probability

Success probability $p$}

\[ |\psi_p\rangle = \sum_p \alpha_p |\psi_p\rangle \]
Our idea

- We use reverse projective property of ProjImp
- Let $D_{\text{fail}}$ be the distribution generates $(b,x,y) \leftarrow D_{\text{wprf}}$ and outputs $(1-b,x,y)$
  - ProjImp($D_{\text{fail}}$) measures failure probability

Success probability $p$
\[ |\psi_p\rangle = \sum_p \alpha_p |\psi_p\rangle \]

Success probability $p$

Failure probability $1-p$

Success probability $p$

Failure probability $1-p$

ProjImp($D_{\text{wprf}}$)

ProjImp($D_{\text{fail}}$)
Our idea

- We use reverse projective property of ProjImp
- Let $D_{\text{fail}}$ be the distribution generates $(b,x,y) \leftarrow D_{\text{wprf}}$ and outputs $(1-b,x,y)$
  - ProjImp($D_{\text{fail}}$) measures failure probability

- ProjImp($D_{\text{wprf}}$) and ProjImp($D_{\text{fail}}$) consist of the same set of operators and the only difference is labels of them
Key fact

Outcome $1/2 + \epsilon$ for some inverse poly $\epsilon$

$\text{ProjImp}(D_{wprf})$

Live
Key fact

Outcome $1/2 + \varepsilon$ for some inverse poly $\varepsilon$

$\text{ProjImp}(D_{wprf})$

$\text{ProjImp}(D_{\text{fail}})$

Live
Key fact

Outcome $1/2 + \varepsilon$ for some inverse poly $\varepsilon$

Live

Outcome $1/2 + \varepsilon$

$\text{ProjImp}(D_{\text{wprf}})$

$\text{ProjImp}(D_{\text{fail}})$
Key fact

Outcome $1/2 + \epsilon$

for some inverse poly $\epsilon$

$\text{ProjImp}(D_{wprf})$

Live

$\text{ProjImp}(D_{\text{fail}})$

Outcome $1/2 - \epsilon$

Outcome $1/2 + \epsilon$
Our extraction method

- Our extraction uses $T_i$ with the following properties for every $i$.

$C' = \text{Mark}(\text{prfk}, m)$
Our extraction method

- Our extraction uses $T_i$ with the following properties for every $i$
Our extraction method

Our extraction uses $T_i$ with the following properties for every $i$
Our extraction method

- Our extraction uses $T_i$ with the following properties for every $i$

  - The outcome of $T_i$ is $1/2+\varepsilon$ if $m_i=0$ and $1/2-\varepsilon$ if $m_i=1$

- From our key fact

  - The outcome of $T_i$ is $1/2+\varepsilon$ if $m_i=0$ and $1/2-\varepsilon$ if $m_i=1$
Our extraction method

- Our extraction uses $T_i$ with the following properties for every $i$
  
- From our key fact
  
  - The outcome of $T_i$ is $1/2 + \varepsilon$ if $m_i=0$ and $1/2 - \varepsilon$ if $m_i=1$
  
  - $T_i$ does not destroy the quantum program
Our extraction method

- Our extraction uses $T_i$ with the following properties for every $i$
  - From our key fact
    - The outcome of $T_i$ is $1/2+\epsilon$ if $m_i=0$ and $1/2-\epsilon$ if $m_i=1$
    - $T_i$ does not destroy the quantum program
  - Our extraction method correctly extracts every bit of $m$ 😊
Summary

1. We define watermarking PRF against quantum adversaries
   - We define unremovability against adversaries who output quantum program

2. We construct watermarking PRF against quantum adversaries
   - Secret extraction scheme based on LWE
   - Public extraction scheme based on IO
     - Our construction methodology is highly general
       and can be extended to watermarking other primitives such as PKE