Different Concepts for Program Obfuscation

Yury Lifshits

Mathematics & Mechanics Faculty
Saint Petersburg State University

Spring 2005 – SETLab
1 Applications

- Classical Cryptography
- Software Protection
- Mobile Agents Technology
- Other
1 Applications
   - Classical Cryptography
   - Software Protection
   - Mobile Agents Technology
   - Other

2 Main Approaches
   - Obfuscating Transformations
   - Blackbox Security
   - Mobile Cryptography
1 Applications
- Classical Cryptography
- Software Protection
- Mobile Agents Technology
- Other

2 Main Approaches
- Obfuscating Transformations
- Blackbox Security
- Mobile Cryptography

3 Aspects of Model
- Program Representation
- Attacks and Environment
Applications for Obfuscation

Yury Lifshits

Applications
Classical Cryptography
Software Protection
Mobile Agents
Technology
Other

Main Approaches
Obfuscating Transformations
Blackbox Security
Mobile Cryptography

Aspects of Model
Program Representation
Attacks and Environment

Summary

Today: only short overview of applications

In details: Lecture 4 - “Applications for Obfuscation”
What applications in cryptography can we imagine?
What applications in cryptography can we imagine?

- Private key cryptosystem → Public key cryptosystem
  It was mentioned even in famous Diffie-Hellman paper.
What applications in cryptography can we imagine?

⇒ Private key cryptosystem → Public key cryptosystem
   It was mentioned even in famous Diffie-Hellman paper.

⇒ Homomorphic encoding
What applications in cryptography can we imagine?

⇒ Private key cryptosystem → Public key cryptosystem  
It was mentioned even in famous Diffie-Hellman paper.
⇒ Homomorphic encoding
⇒ Random oracles removing
Situation: we *distribute (sell)* software products.

**Question**: Threats and applications you see?
Situation: we distribute (sell) software products.

**Question**: Threats and applications you see?

⇒ Competitors threat (reusing your code)
Situation: we distribute (sell) software products.

**Question**: Threats and applications you see?

- Competitors threat (reusing your code)
- Intelligent tampering (changing parameters)
Situation: we distribute (sell) software products.

**Question**: Threats and applications you see?

- Competitors threat (reusing your code)
- Intelligent tampering (changing parameters)
- Threat of functionality changes (protection demo-versions)
Situation: we **distribute (sell)** software products.

**Question**: Threats and applications you see?

- Competitors threat (reusing your code)
- Intelligent tampering (changing parameters)
- Threat of functionality changes (protection demo-versions)
- Watermarks protection
Situation: we distribute programs for our needs.

Question: Threats and applications you see?
Situation: we distribute programs for our needs.

**Question**: Threats and applications you see?

⇒ Privacy: e.g. internet-distributed computation
Situation: we distribute programs for our needs.

**Question**: Threats and applications you see?

- Privacy: e.g. internet-distributed computation
- Keys protection: buying agents.
Situation: we distribute programs for our needs.

**Question**: Threats and applications you see?

- Privacy: e.g. internet-distributed computation
- Keys protection: buying agents.
- Intelligent tampering
**Question**: More applications?
**Question**: More applications?

Yes!
Question: More applications?
Yes!

⇒ Virus development
Question: More applications?

Yes!

- Virus development
- Watermark attacks
In details: Lecture 2 - “Obfuscating transformations”

Program $P_{\text{clear}}$ → Obfuscator → Obfuscated $P_{\text{unreadable}}$
In details: Lecture 2 - “Obfuscating transformations”

- Functionality preserving
- Increase of code size, time & space requirements are restricted (usually by constant factor)
- Obfuscated program is not readable (not understandable)
What can we obfuscate in the program?
What can we obfuscate in the program?

⇒ Layout transformations
   Change formatting information
What can we obfuscate in the program?

- Layout transformations
  Change formatting information
- Control flow transformations
  Alter control program and computation
What can we obfuscate in the program?

⇒ Layout transformations
  Change formatting information

⇒ Control flow transformations
  Alter control program and computation

⇒ Aggregation transformation
  Refactor program using aggregation methods
Classification of obfuscating transformations

What can we obfuscate in the program?

- **Layout transformations**  
  Change formatting information

- **Control flow transformations**  
  Alter control program and computation

- **Aggregation transformation**  
  Refactor program using aggregation methods

- **Data transformations**  
  Use information encoding
Quality of Obfuscation

How good our obfuscation is?
How good our obfuscation is?

Strength can be measured by:

⇒ Potency
\[ \frac{E(P')}{E(P)} - 1 \]

⇒ Resilience
Trivial, weak, strong, full, one-way

⇒ Cost
Free, cheap, costly, expensive

⇒ Stealthy
What do we want to get?

Ideal case
What do we want to get?

```c
mysterious.c

int mysterious(int x, int y)
{
    int z;
    z = x + y;
    return z;
}
```
What do we want to get?

Very limited information:
- input-output behavior
- running time
We are interested in 2 types of polynomial-time analyzers:

- **Ana** is a source-code analyzer that can read the program.
  \[ \text{Ana}(P) \]

- **BAna** is a black-box analyzer that only queries the program as an oracle.
  \[ B\text{Ana}^P(\text{time}(P)) \]
We are interested in 2 types of polynomial-time analyzers:

- **Ana** is a source-code analyzer that can read the program.
  \[
  \text{Ana}(P)
  \]

- **BAna** is a black-box analyzer that only queries the program as an oracle.
  \[
  \text{BAna}^P\left(\text{time}(P)\right)
  \]

**Black-Box security**

Ana can’t get more information than BAna could
How to formalize property hiding?
How to formalize property hiding?

Instance: two families of programs $\Pi_1$ and $\Pi_2$

**Adversary task**: given a program $P \in \Pi_1 \cup \Pi_2$ to decide whether $P \in \Pi_1$ or $P \in \Pi_2$. 
How to formalize property hiding?

**Instance:** two families of programs $\Pi_1$ and $\Pi_2$

**Adversary task:** given a program $P \in \Pi_1 \cup \Pi_2$ to decide whether $P \in \Pi_1$ or $P \in \Pi_2$.

**Desirable protection:** make adversary task as difficult as well-known computationally hard problem is.
How to formalize constant hiding?
How to formalize constant hiding?

Instance: family of programs

$\Pi = \{ P | P \text{ computes } f(s, x); \ s \in S \}$

Adversary task: given a program $P \in \Pi$ to compute parameter $s$. 
How to formalize constant hiding?

Instance: family of programs

\[ \Pi = \{ P | P \text{ computes } f(s, x); \ s \in S \} \]

Adversary task: given a program \( P \in \Pi \) to compute parameter \( s \).

Desirable protection: make adversary task as difficult as well-known computationally hard problem is.
More details: Lecture 5 - “Basic Complexity Results”

**What is encrypted computation?**

Basic task: keep $F$ unknown to Bob.
Additional tasks of encrypted computation model:

- Move difficult computations to Bob
  - $D$ is easier than $F$
- Reduce communication complexity
  - In the case $\text{sizeof}(F(x)) \ll \text{sizeof}(x)$. Example: $x$ is database
- Keep $x$ secret from Alice
Currently studied representations

Obfuscating techniques development depends on used program representation

So what sort of programs are we going to protect?
Obfuscating techniques development depends on used program representation

So what sort of programs are we going to protect?

⇒ Turing Machines / Circuits (function computing)
Obfuscating techniques development depends on used program representation

So what sort of programs are we going to protect?

- Turing Machines / Circuits (function computing)
- C++/Java code
Obfuscating techniques development depends on used program representation

So what sort of programs are we going to protect?

⇒ Turing Machines / Circuits (function computing)
⇒ C++/Java code
⇒ Assembler code
Currently studied representations

Obfuscating techniques development depends on used program representation

So what sort of programs are we going to protect?

- Turing Machines / Circuits (function computing)
- C++/Java code
- Assembler code
- Rational function / Matrix representation
Is it enough?
Search for other representations

Is it enough?

Not! New models should contain:
Is it enough?

Not! New models should contain:

⇒ Current state of the program.
Search for other representations

Is it enough?

Not! New models should contain:

⇒ Current state of the program.
⇒ Self-modifiable code
Is it enough?

Not! New models should contain:

⇒ Current state of the program.
⇒ Self-modifiable code
⇒ Notion of computation trace.
Is it enough?

Not! New models should contain:

- Current state of the program.
- Self-modifiable code
- Notion of computation trace
- Other formalizations for functionality preserving
What should we specify about adversary?
What should we specify about adversary?

⇒ Adversary knowledge about protected program
What should we specify about adversary?

⇒ Adversary knowledge about protected program
   ■ Member of family
What should we specify about adversary?

- Adversary knowledge about protected program
  - Member of family
  - Known function – unknown parameters (data) and state.
What should we specify about adversary?

- Adversary knowledge about protected program
  - Member of family
  - Known function – unknown parameters (data) and state.
- Adversary task (attack)
What should we specify about adversary?

⇒ Adversary knowledge about protected program
  ■ Member of family
  ■ Known function – unknown parameters (data) and state.

⇒ Adversary task (attack)
  ■ Classification follows in Lecture 4.
Is it possible to protect every program?
Is it possible to protect every program?

⇒ How to measure potential of obfuscation?
- Learnability: black-box learnable functions are impossible to obfuscate.

⇒ What couldn’t be protected?
- Input-Output behaviour
- Traces
What are interesting network extensions of the model?
What are interesting network extensions of the model?

- Many programs cooperate
- Programs are migrating
- Programs can be recharged
- Different sources for inputs (outside connections)
Rough idea of applications: cryptosystem design, mobile agents technology, software protection.
⇒ Rough idea of applications: cryptosystem design, mobile agents technology, software protection.
⇒ Basic approaches: obfuscating transformations, black-box security, encrypted computation.
Rough idea of **applications**: cryptosystem design, mobile agents technology, software protection.

Basic **approaches**: obfuscating transformations, black-box security, encrypted computation.

Further **aspects** of the model: program representation, state protection, adversary description, functionality preserving.
Rough idea of applications: cryptosystem design, mobile agents technology, software protection.

Basic approaches: obfuscating transformations, black-box security, encrypted computation.

Further aspects of the model: program representation, state protection, adversary description, functionality preserving.
Rough idea of applications: cryptosystem design, mobile agents technology, software protection.

Basic approaches: obfuscating transformations, black-box security, encrypted computation.

Further aspects of the model: program representation, state protection, adversary description, functionality preserving.

Question Time!
Gray & white security
Approximate obfuscators
Operations on obfuscated code
Adversary success
Nondeterministic nature
Modifying algorithm vs. modifying code
Complexity of deobfuscation: NP, NP-hard, undecidable, one-way...
Obfuscation on specification level
Wroblewsky model
Yury Lifshits

Program Obfuscation. A Survey [in Russian]


Luis F.G. Sarmenta

Protecting Programs from Hostile Environments

http://bayanihancomputing.net/papers/ae/ae.ps