Modern trends in discrete mathematics and computer science

Chair of Mathematical Cybernetics,
Faculty of Computational Mathematics and Cybernetics,
Lomonosov Moscow State University

A brief outline of the course

1. Theory and practice of software obfuscation (V.A. Zakharov)

2. Formal correctness proofs for sequential programs

(V.V. Podymov)

3-4. Constraint satisfaction problem: an algebraic approach

(S.N. Selezneva)

5. Read-once functions

(A.A. Voronenko)

A brief outline of the course

6. Closed classes in k-valued and partial k-valued logics

(V.B. Alekseev)

7. Some new bounds of Shannon functions for cardinality of test sets for logical circuits

(D.S. Romanov)

8. Mathematical models in 13C metabolic flux analysis

(M.S. Shupletsov)

Theory and practice of software obfuscation

Vladimir A. Zakharov

Lomonosov Moscow State University,
Faculty of Computational Mathematics and Cybernetics

1. What is a software obfuscation?

2. What is obfuscation good for?

3. How to obfuscate programs in practice?

4. How to guarantee security of software obfuscation?

WHAT IS AN OBFUSCATION OF PROGRAMS?

Origin

The word *obfuscation* derives from the Latin obfuscare (to make dark, to obscure).

Obfuscation is the obscuring of the intended meaning of communication by making the message difficult to understand.

Software obfuscation (quite informally):

An obfuscation of computer programs is any semantic-preserving transformation O which brings an arbitrary program π into such a form $O(\pi)$ which is a “virtual black-box”.

Just a little more formally:

Obfuscation is any transformation of programs which complies with the following requirements:

1. It preserves the semantics of a source program;

2. It does not deteriorate considerably both the size and the efficiency (performance) of a program;

3. Deducing any useful information about the implemented algorithms and data structures from the plaintext (code) of an obfuscated program, as well as making its purposeful modification becomes a far more hard task.

Some History:

Program Obfuscation was pioneered in

Diffie W., Hellman M.E., New directions in cryptography. IEEE Transactions in Information Theory, 1976.

But the computer science community got to know this problem from

Collberg C., Thomborson C., Low D., A taxonomy of obfuscating transformations, Tech. Report, N 148, Dept. of Computer Science, University of Auckland, 1997.

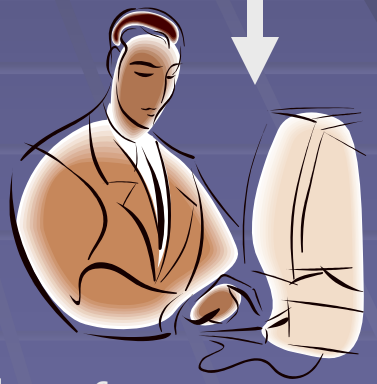
Mathematical Cryptology
Complexity Theory
Program Obfuscation
Program Analysis
Reverse Engineering

WHAT OBFUSCATION IS GOOD FOR?

To protect software intellectual property

Alice, software engineer

Executable code



Bob, software user

To protect software intellectual property

Alice, software engineer

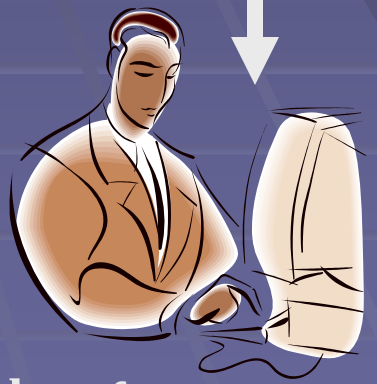
Executable code

Harry,
malicious hacker

Algorithms and
data structures

Source code

Decompiling tool

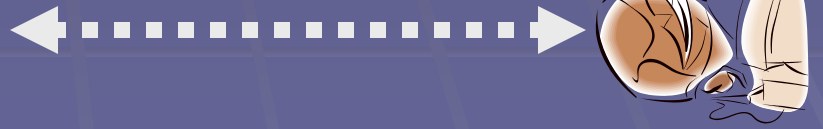


Bob, software user

To protect software intellectual property

Alice, software engineer

Executable code



Bob, software user

MACROHARD Inc.

To protect software intellectual property

Alice, software engineer

Source code

OBFUSCATING COMPILER

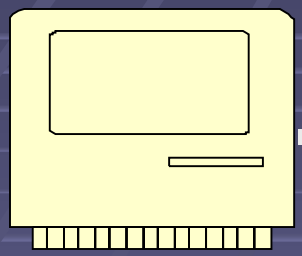
Obfuscated executable code

To protect programs from unauthorized modification

Program obfuscation is also applicable for providing security of

- mobile agents operating in hostile environment;
- project designs in manufacturing VLSI circuits;
- cloud computing.

Designing public key cryptosystems



Private key cryptosystem (e.g., DES, AES, etc.)

Encryption program

Encryption program adapted to a fixed private key

OBFUSCATOR

Obfuscated encryption program

Public key

Private key

Key

Key

Private key

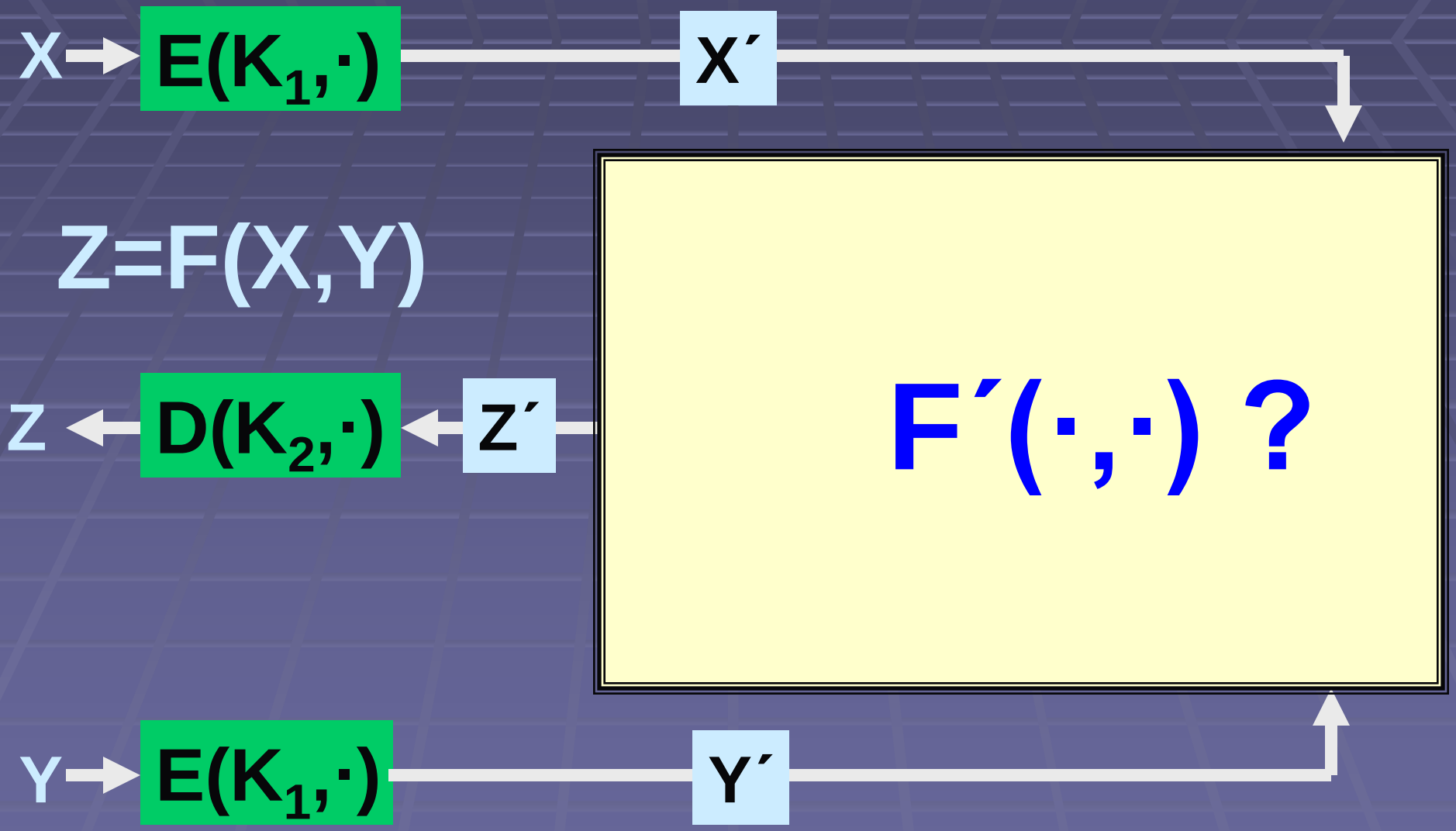
Designing homomorphic cryptosystems

A cryptosystem is called homomorphic if it provides a possibility to perform arithmetic and Boolean operations on encrypted data:

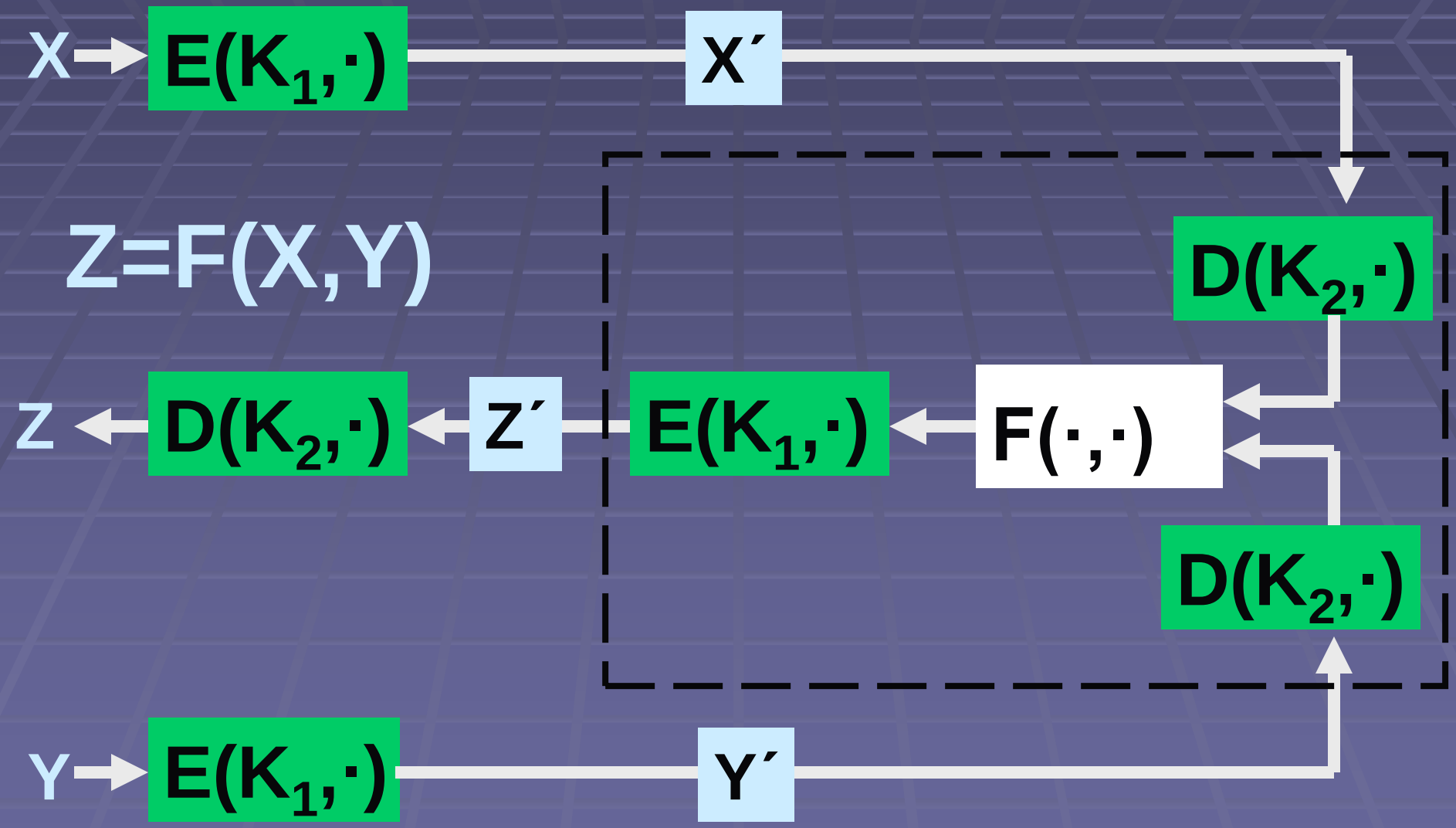
$$E(X \oplus Y) = E(X) \otimes E(Y)$$

E.g. $q^{x+y} = q^x \times q^y$ (El Gamal encryption)

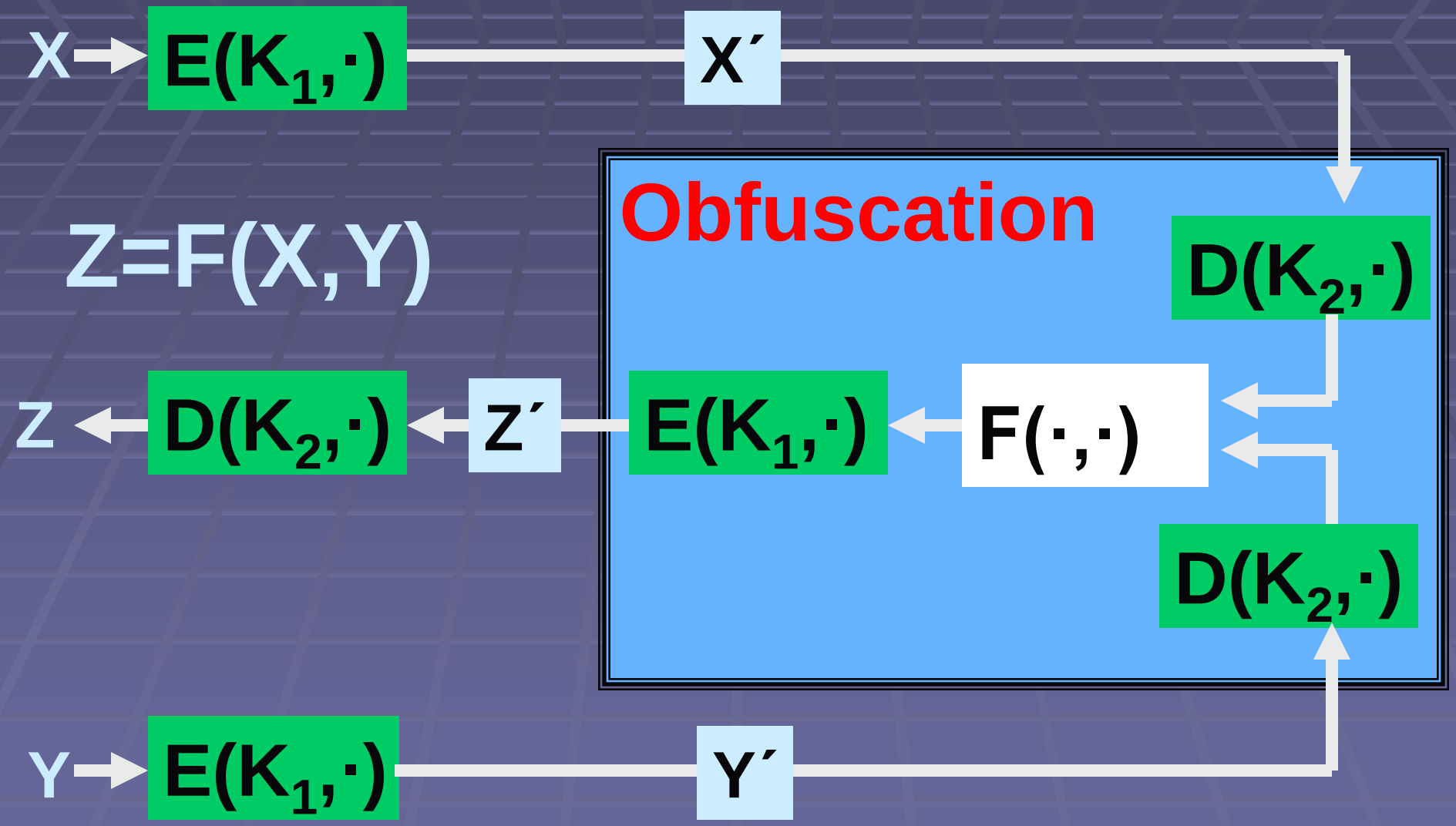
Designing homomorphic cryptosystems



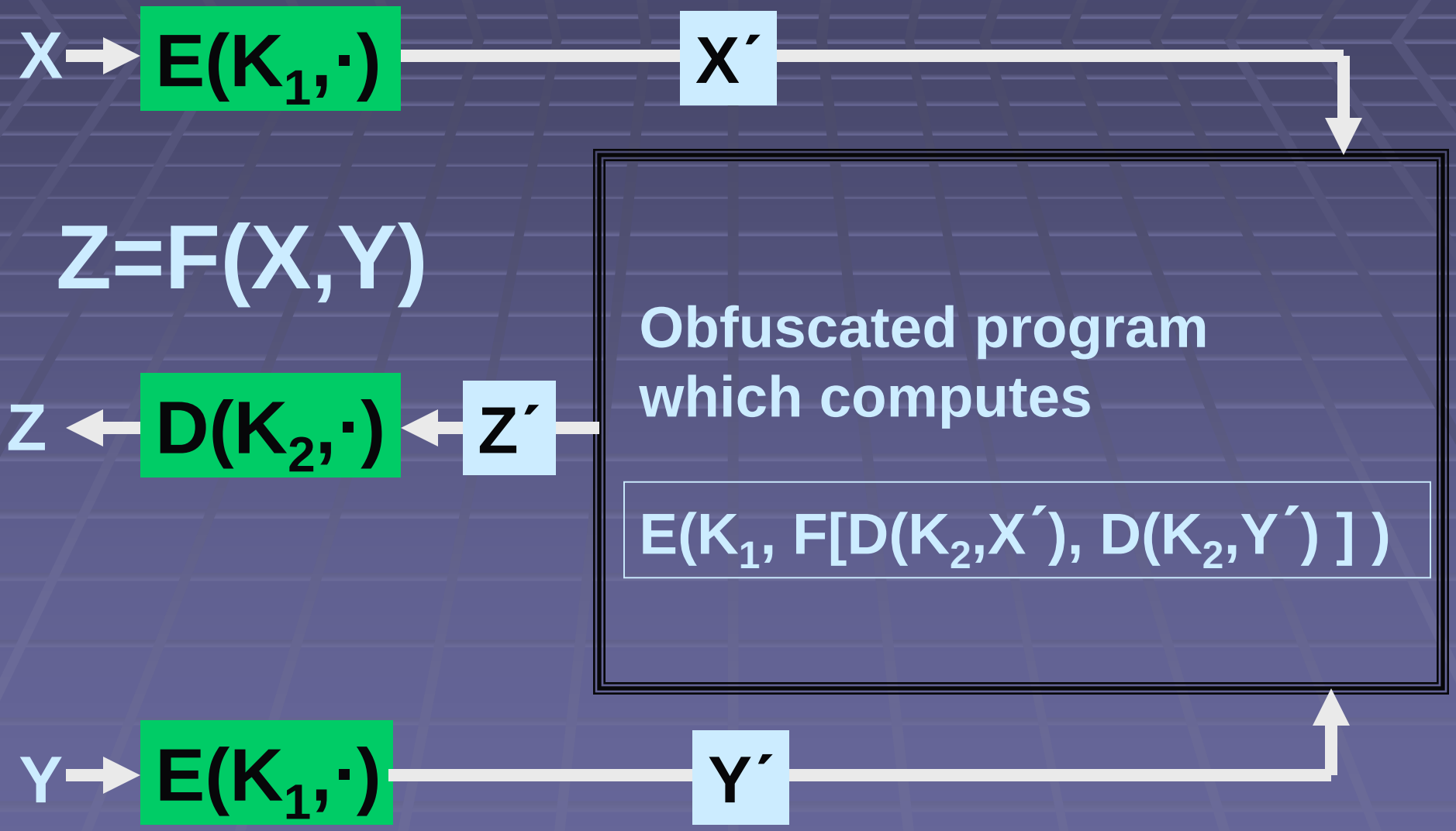
Designing homomorphic cryptosystems



Designing homomorphic cryptosystems



Designing homomorphic cryptosystems



Other applications of obfuscation

- Hiding unspecified functionalities (bugs) or vulnerabilities in computer programs;
- Designing metamorphic malwares;
- Designing “watermarks” and “fingerprints”;
- Combining privacy and publicity in elections;
- Providing privacy in information searching

HOW TO OBFUSCATE PROGRAMS?

Some positive arguments in favour of software obfuscation

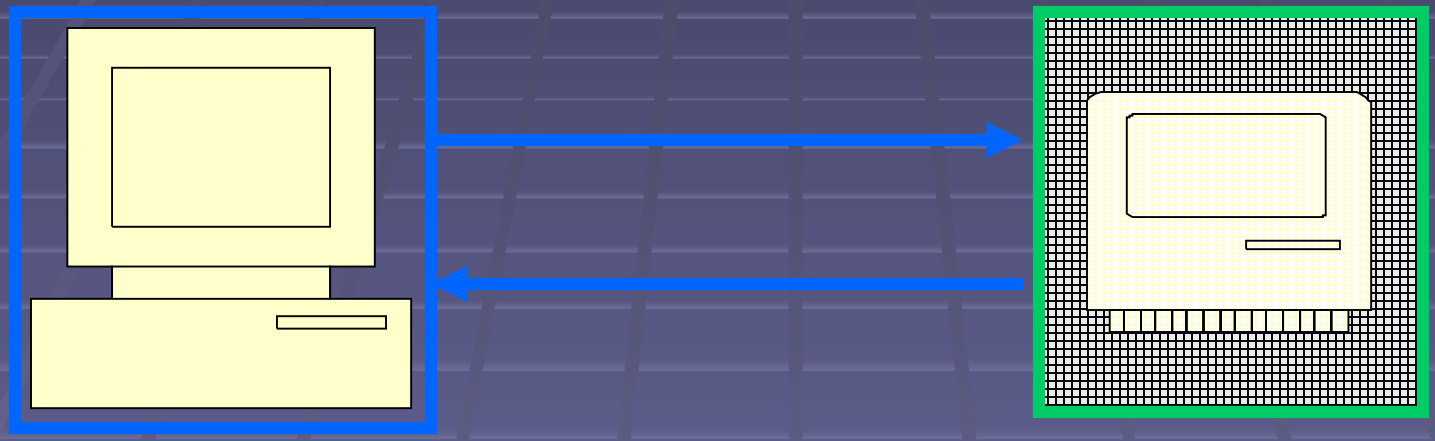
1. Formal analysis of programs is a very hard task (see, e.g. Rice's Theorem).

2. Even the best software designers can't comprehend completely a behaviour of programs they developed.

The principal approaches to program obfuscation

1. Program obfuscation with the using of a secure computing unit.
2. “Pure” obfuscation.

Program obfuscation with the using of secure computer

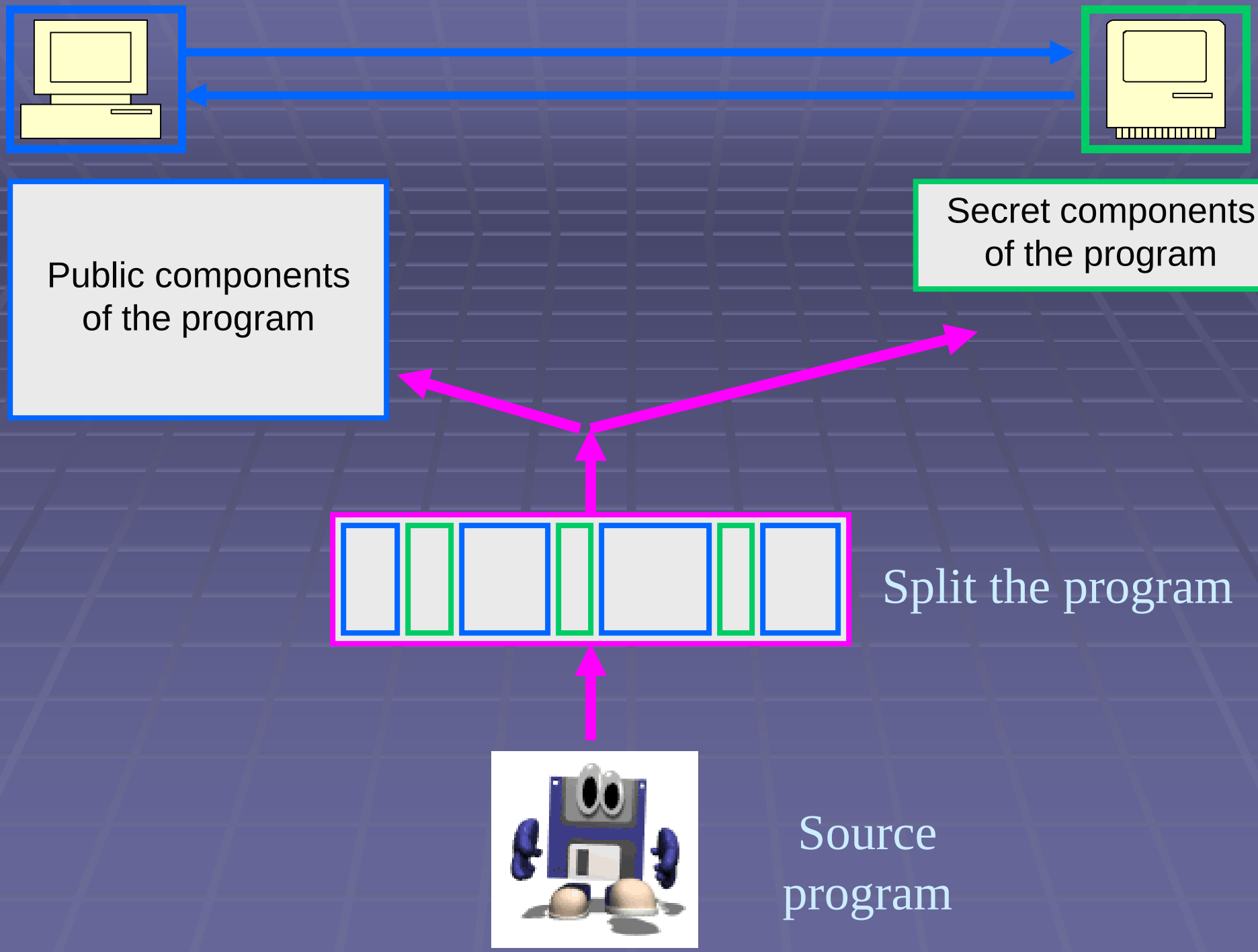


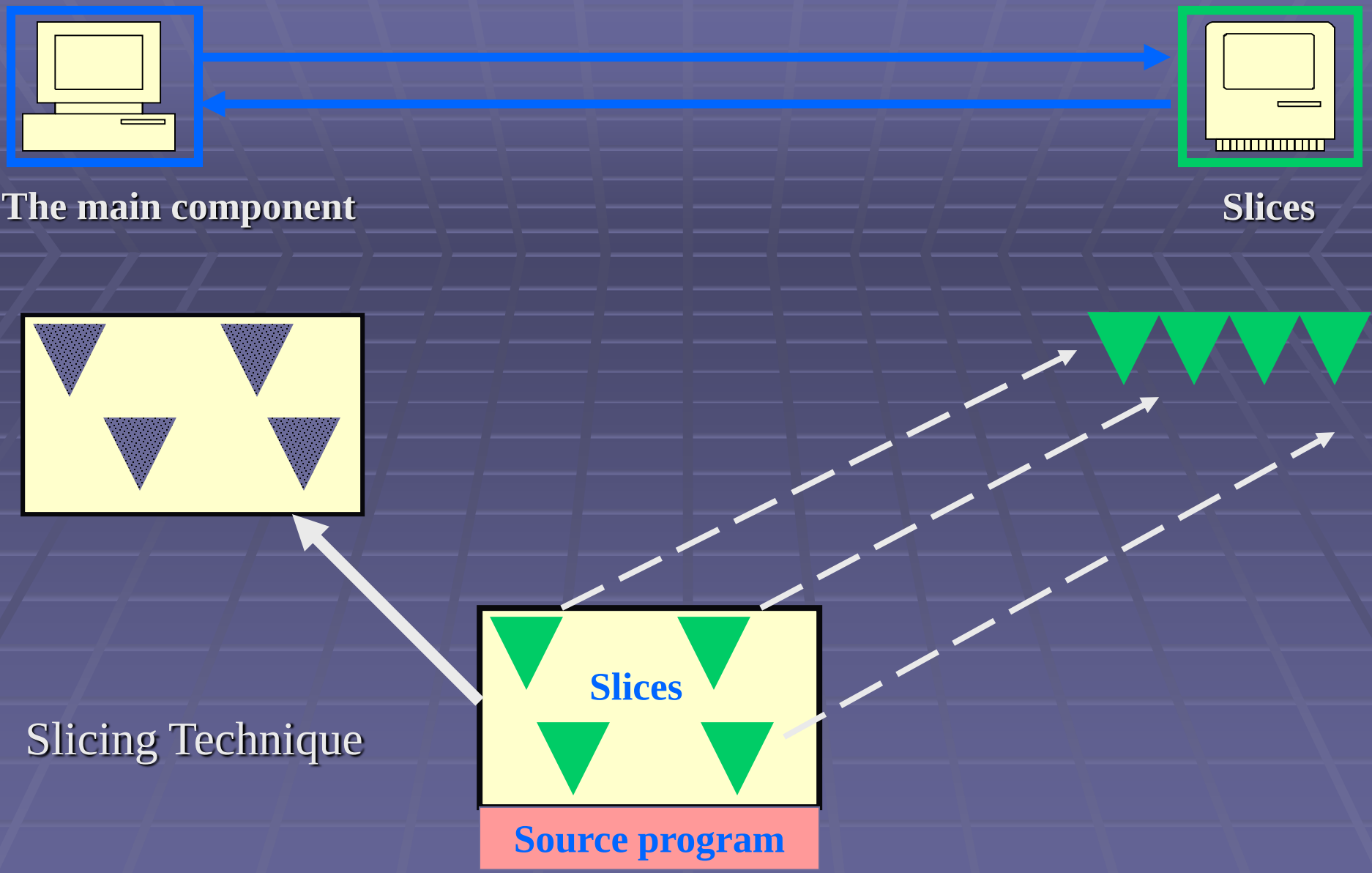
High-performance computer open to general use

(e.g. a bank server)

Secure low-performance computing unit

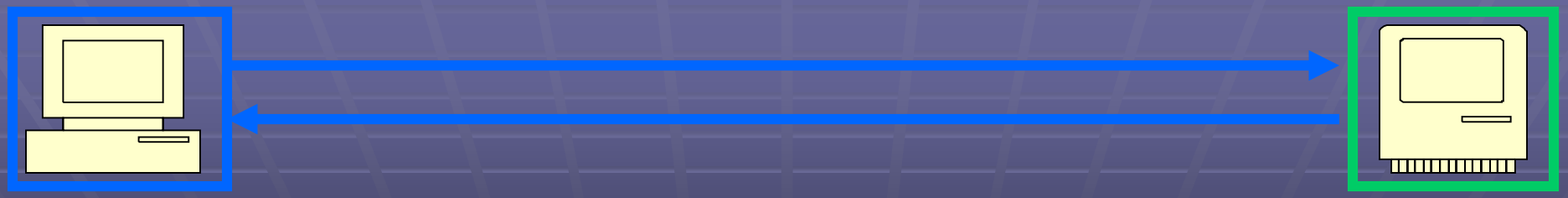
(e.g. smartcard of a client)


Public components of the program
Secret components of the program
Split the program
Source program



Slicing Technique

X. Zhang, R. Gupta, Y. Zhang, Hiding Program Slices for Software Security, ***First Annual IEEE/ACM International Symposium on Code Generation and Optimization***, 2003.



Open memory unit

Secure processor

If one-way functions exist then every program π can be brought into such a form Π that:

1. $\text{Time}(\Pi) = \text{Time}(\pi) * \log^3(\text{Time}(\pi))$;

2. When executing Π on a computing system which is composed of a secure processor and unsecure (public) memory unit no adversary can in polynomial time distinguish the program Π by a sequence of its storage accesses.

R. Ostrovsky, Efficient computation on oblivious RAM,
Proc. of 22nd ACM Symposium on Theory of Computing (STOC-90)

Computation on encrypted data (homomorphic encryption)

Given a program **A** choose a pair **(E,D)** of encoding and decoding algorithms and transform **A** into a program **B** such that

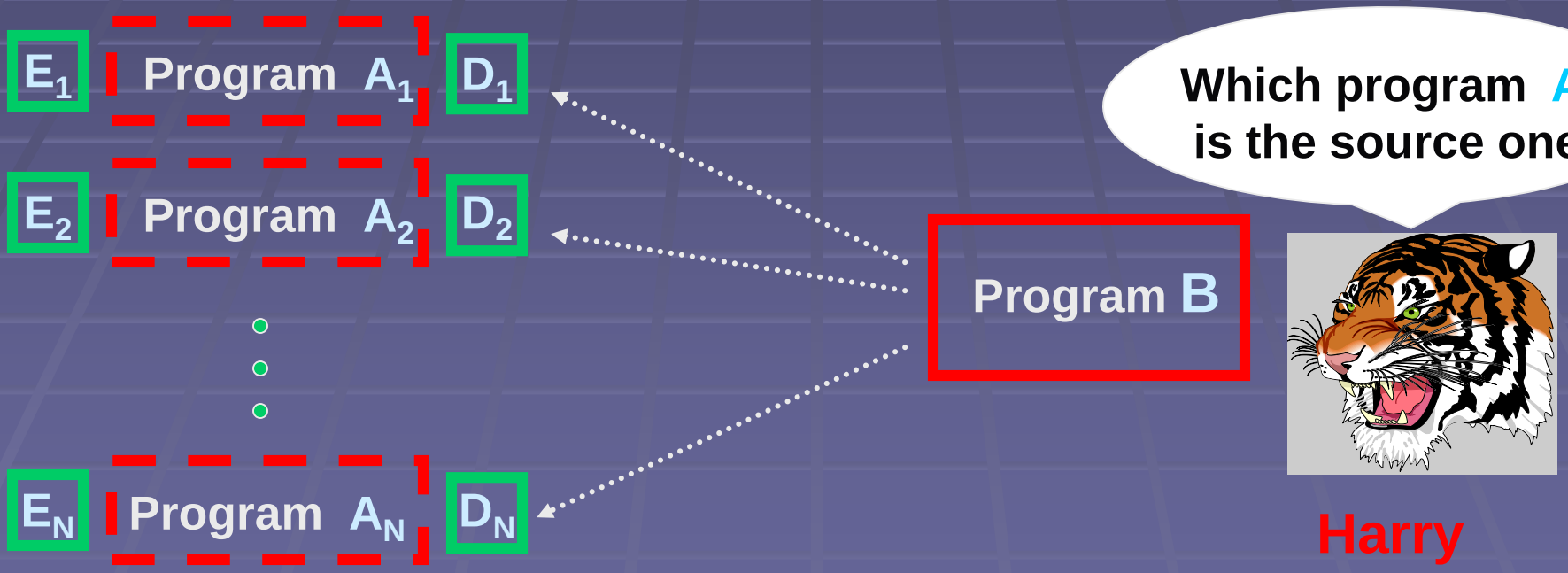
$$A(x) = D\,(B\,(E(x))\,)$$

holds for any input **x**

Computation on encrypted data (homomorphic encryption)

Security of program obfuscation with the help of data encryption is guaranteed by the fact that a wide variety of possible triples **(A′, E′, D′)** correspond to the same visible program **B**.

Thus, **B** gives an adversary no intelligence of the source program **A**.



Computation on encrypted data (homomorphic encryption)

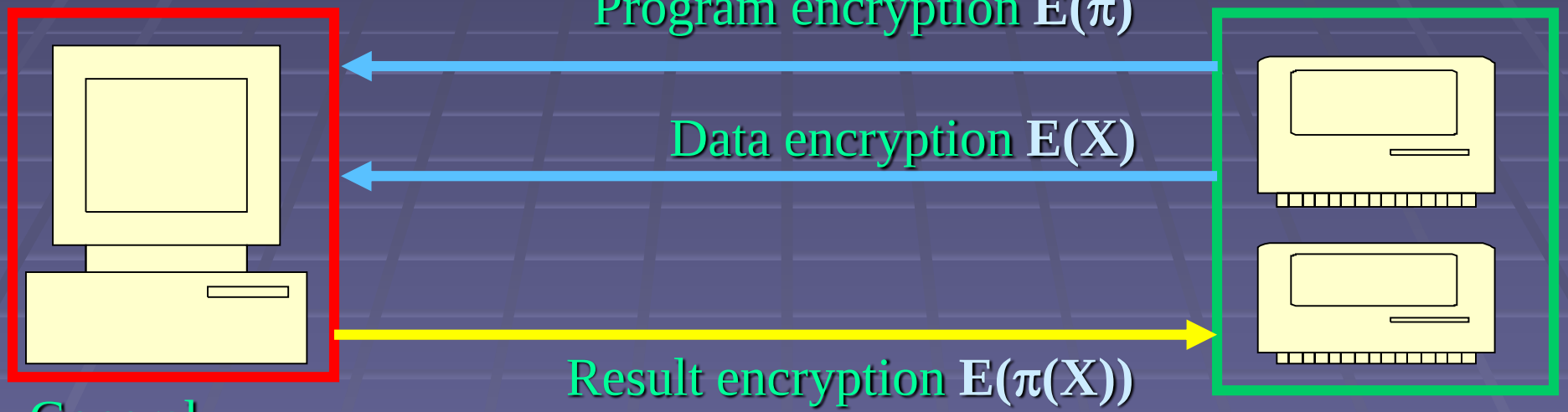
Theorem. There exists a compiler ***O*** which transforms any loop-free program ***A*** implementing algebraic operations into a system ***(E,D,B)*** such that:

- The size of ***B*** is almost the same as that of ***A***;
- The number of programs ***A´*** that correspond to ***B*** for some appropriate ***E´,D´*** is exponential of the size of ***B***.

T. Sander, C. Tchudin, On software protection via function hiding,
Proc. of the 2-nd Workshop on Inf. Hiding, 1998.
A.V. Shokurov, An approach to quantitative analysis of resistance of data encodings in tamper-resistant software,
Tech. Reports of the Institute for system programming, 2004.

Fully homomorphic encryption

Craig Gentry. "A Fully Homomorphic Encryption Scheme". Ph.D. Thesis, MIT, 2009.



$$E(\pi[X]) = E(\pi)[E(X)]$$

Fully homomorphic encryption

Key Generation: choose an arbitrary odd integer p

Encryption of a binary m:

- choose an arbitrary large enough integer q and some rather small integer r ,
- compute $c = pq + 2r + m$.

Decryption of a ciphertext c:

- compute $m = (c \bmod p) \bmod 2$.

Fully homomorphic encryption

Suppose that $c_1=q_1p+2r_1+m_1$, $c_2=q_2p+2r_2+m_2$

Then $c_1+c_2 = (q_1+q_2)p + 2(r_1+r_2) + (m_1+m_2)$

If $2(r_1+r_2)+(m_1+m_2)$ is far less than p, then

$Dec(Enc(m_1) + Enc(m_2)) =$

$(c_1+c_2 \bmod p) \bmod 2 =$

$(2(r_1+r_2) + (m_1+m_2)) \bmod 2 =$

$m_1+m_2 \bmod 2 = m_1 \oplus m_2$

Therefore, $Dec(Enc(m_1)+ Enc(m_2)) = m_1 \oplus m_2$

Fully homomorphic encryption

Suppose that $c_1 = q_1 p + 2r_1 + m_1$, $c_2 = q_2 p + 2r_2 + m_2$

Then $c_1 \times c_2 = (c_1 q_2 + q_1 c_2 - q_1 q_2)p + 2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 m_2$

If $2(2r_1 r_2 + r_1 m_2 + m_1 r_2)$ is far less than p, then

$Dec(Enc(m_1) \times Enc(m_2)) =$

$(c_1 \times c_2 \bmod p) \bmod 2 =$

$(2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 m_2) \bmod 2 =$

$m_1 \times m_2$

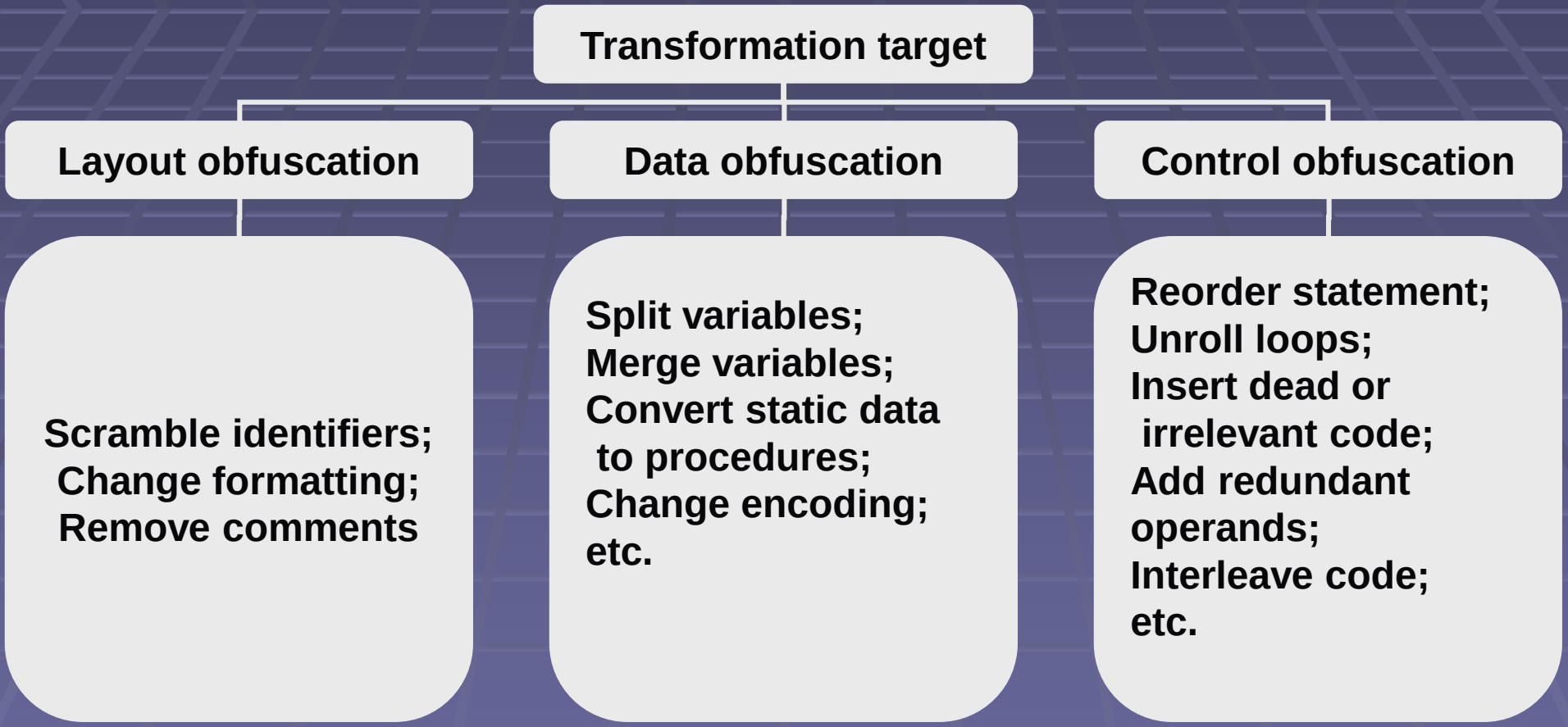
Therefore, $Dec(Enc(m_1) \times Enc(m_2)) = m_1 \wedge m_2$

“Pure” obfuscation

Unlike obfuscation with the using of a secure computing unit, pure obfuscation assumes that an adversary is granted unlimited access to the whole program (cf. private-key vs. public-key cryptography).

C. Collberg, C. Thomborson, D. Low, A taxonomy of obfuscating transformations,
Tech. Report, N 148, Dept. of Computer Science, Univ. of Auckland, 1997.

A Taxonomy of Obfuscating Transformations



(renaming, dynamic data structures, useless and dead code, opaque predicates)

```
function fib
(var n:int):int;
var k,m: int;
m=1; k=1;
while n > 2
 do
 m=m+k;
k=m-k;
n=n-1
 od;
return(m);
```

```
function T7 (var l10,llO:int):int;
var ll0,l1O,ll,l1: longint;
ll0=1; ll=(l10+1)*(l10-1)/l10O; l1=2;
l1O=ll0+sin(7.2638*(ll-l10))+l10;
while (llO –l1)(9*l10*(l10-1)+2) > 0
  do
     ll=l1O-l10; ll0=ll0+ll;
if llO < l1 then l10=l10*ll else l10=l10-ll
    fi
      l1O=ll0+l10; l1=l1+1;
  od;
return(ll0);
```

Opaque predicates

A predicate P is called opaque if its behaviour is known at the stage of compilation but it is hard to deduce this behaviour by analyzing the obfuscated program:

say, instead of writing a simple assignment

```
b=1
```

one can write a compound statement

```
if (x**2+1)**3+(y**2+1)**3=(z**2+1)**3
   then  b=0
   else  b=1 fi
```

Opaque predicates

Some combinatorial problems are well suited for building opaque predicates.

For example, given a simple Boolean formula

$$x_1 \wedge x_2$$

one could transform it into an expression

$$(x_3 \vee \overline{x_4} \vee x_1) \wedge (x_4 \vee x_2) \wedge (\overline{x_4} \vee x_2)$$

$$\wedge (x_1 \vee x_4) \wedge (\overline{x_3} \vee x_1)$$

which is less easy for understanding

Opaque predicates

Some combinatorial problems are well suited for building opaque predicates.

For example, given a simple Boolean formula

$$x_1 \wedge x_2$$

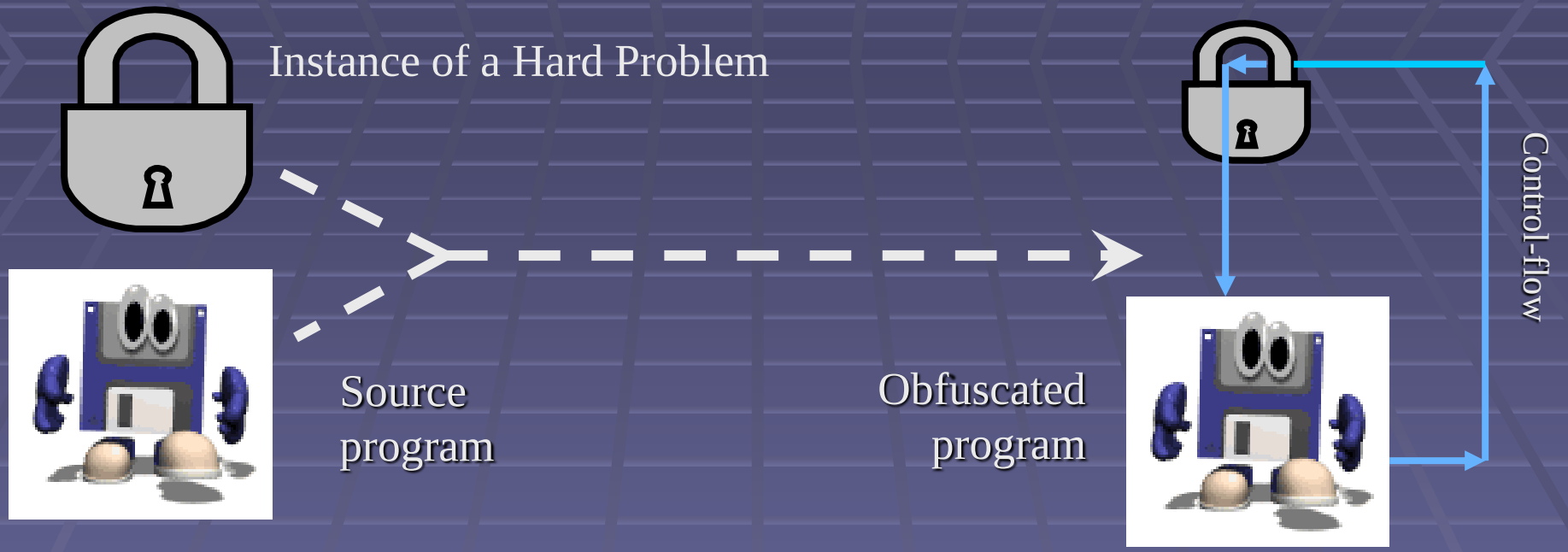
one could transform it into an expression

$$(x_3 \vee \overline{x_4} \vee x_1) \wedge (x_4 \vee x_2) \wedge (\overline{x_4} \vee x_2)$$

$$\wedge (x_1 \vee x_4) \wedge (\overline{x_3} \vee x_1)$$

which is less easy for understanding

Implantation of a computationally hard problem into a program

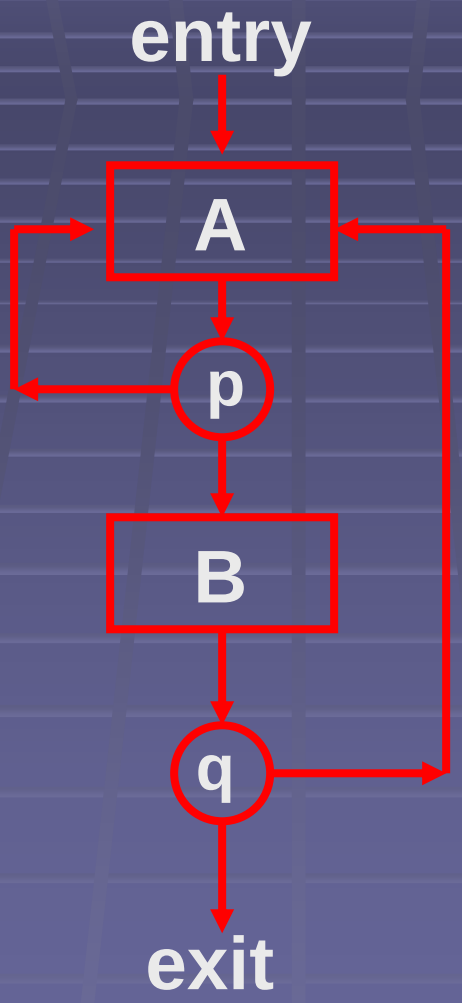


Wang C., Hill J., Knight J. Davidson J., Software tamper resistance:obstructing static analysis of programs, *Tech. Rep., N 12, Dep. Of Comp. Sci., Univ. of Virginia*, 2000.

Chow S., Gu Y., Johnson H. Zakharov V., An approach to the obfuscation of control flow of sequential computer programs. Information Security Conference, 2001.

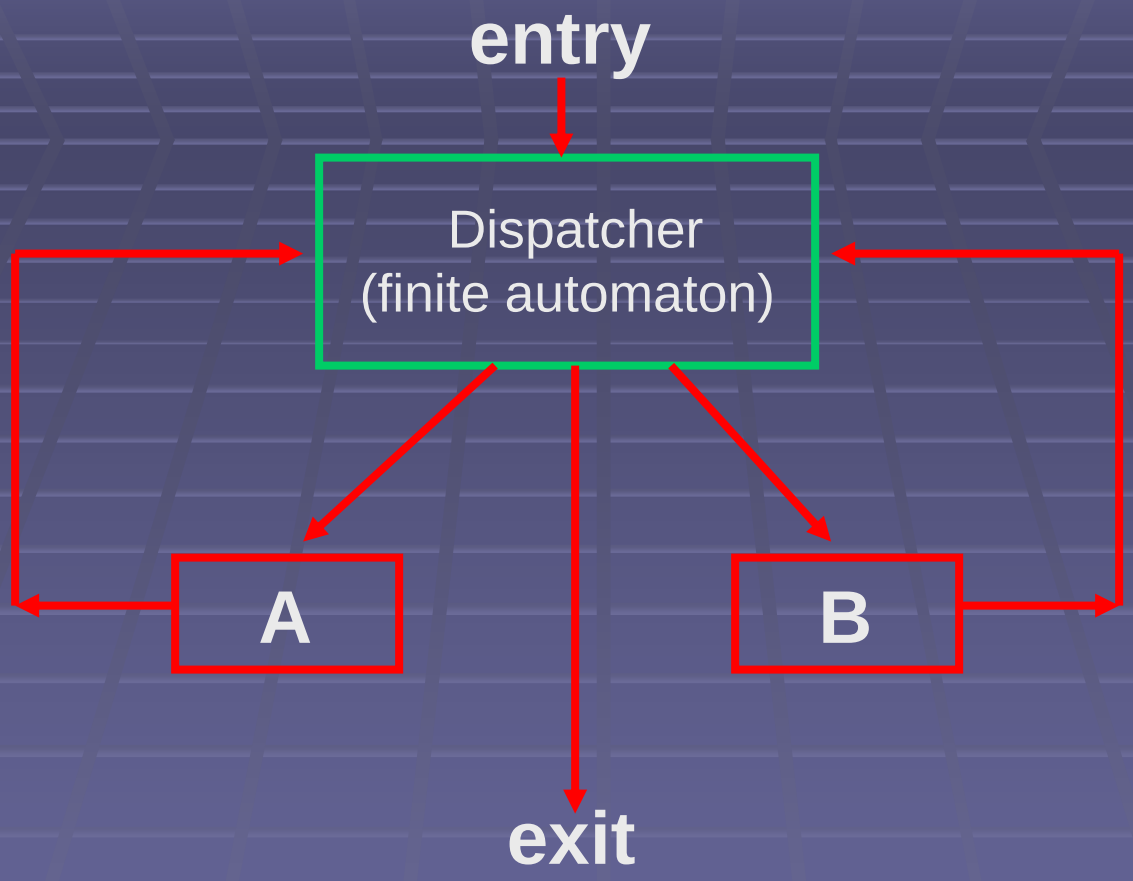
T. Ogiso, Y. Sakabe, M. Soshi, Miyaji. Software obfuscation on a theoretical basis and its implementation. ***IEICE Trans. on Fundamentals***, 2003.

Obfuscation through the “flattening” of a program



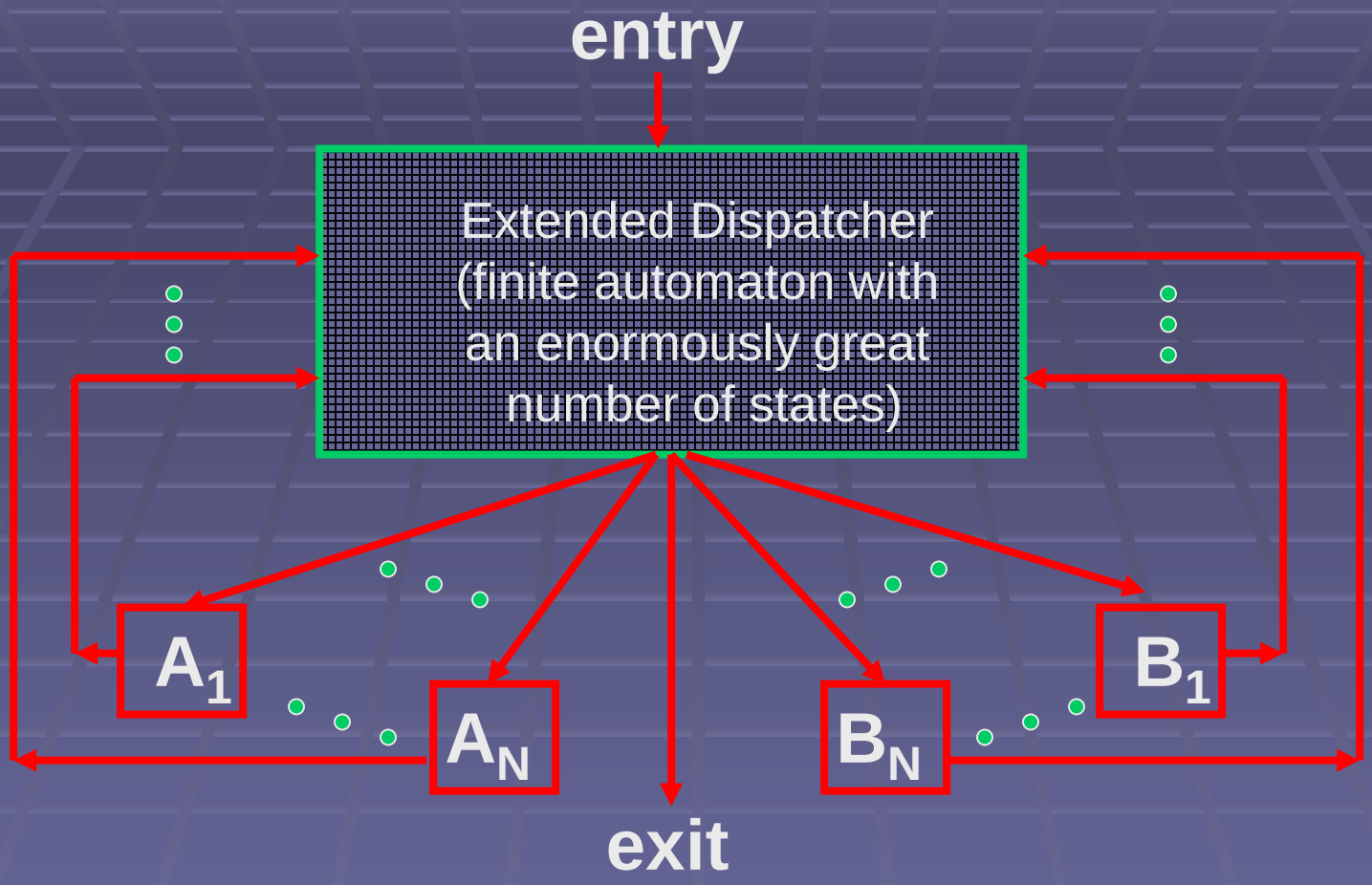
Source program

Obfuscation through the “flattening” of a program



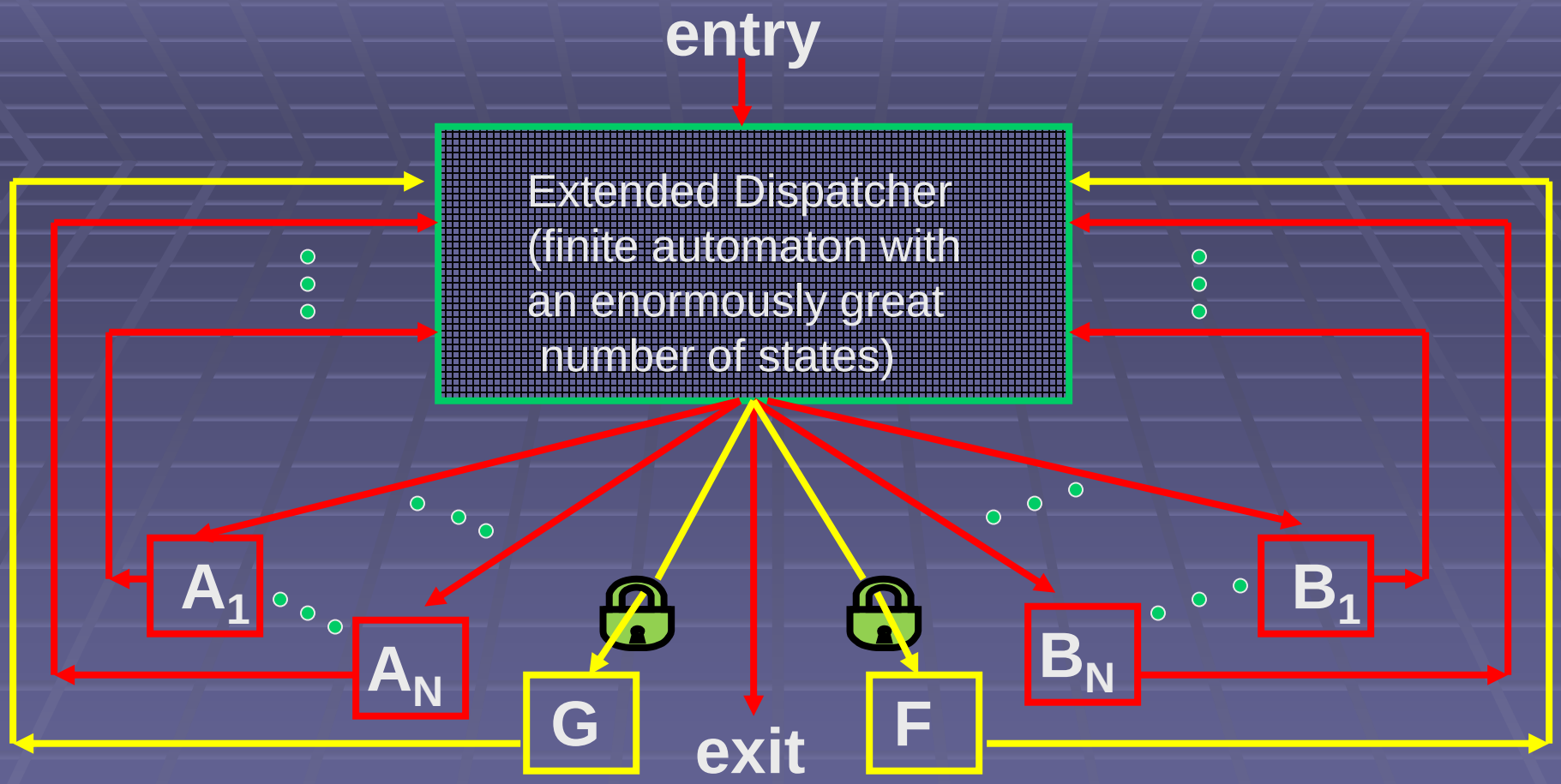
The “flattening” is achieved by introducing a dispatcher which accumulates the control-flow of a program.

Obfuscation through the “flattening” of a program



Cloning and modifying basic blocks.

Obfuscation through the “flattening” of a program



Adding fake basic block supplied with ”opaque” predicates.

HOW TO GUARANTEE OBFUSCATION SECURITY?

Obfuscation and Cryptography

The similarity:

- Cryptography: $C=E(m)$

  C is a ciphertext of a message m.

- Obfuscation: $\Pi = O(\pi)$

  Π is a "cryptogram" of a source program π.

The main difference:

An obfuscated program ("cryptogram") Π must have the same meaning (semantics) as a source program π. Therefore obfuscation may be thought of as a semantic-preserving encryption of programs.

A “black box” obfuscation

An obfuscator O is called universal and secure if for every program π an obfuscated program $O(\pi)$ is a “virtual black box”, i.e. every algorithm A when analyzing a program $O(\pi)$, achieves in polynomial time no better results than some testing algorithm S, which is applied to $O(\pi)$ as to a “black box”.

“virtual black box” property:

for every Probabilistic Polynomial Time Turing Machine (PPT) A there exists a PPT S and a negligibly small function $neg(\cdot)$ such that for all programs π

$$|Pr[A(O(\pi))=1] - Pr[S^{\pi}(1^{|\pi|})=1]| \leq neg(|\pi|),$$

On the impossibility of secure "black box" obfuscation

Theorem. Universal secure obfuscators do not exist.

Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vedhan S, Yang K., On the (Im)possibility of obfuscating programs. ***CRYPTO'01 - Advances in Cryptology, 2001.***

Goldwasser S., Kalai Y.T. On the impossibility of obfuscation with auxiliary inputs. ***Proc. FOCS 2005.***

On the impossibility of secure obfuscation

Singular (point) function:

$f_{a,b}(x)$: **if** (x=a) **then** return(b) **else** return(0).

Singular checker:

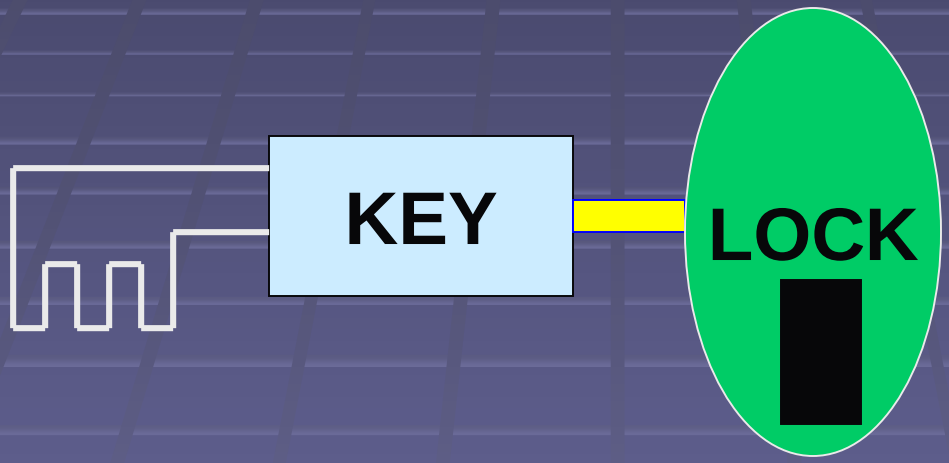
$Ch_{a,b}$ (x): **if** (x(a)=b) **then** return(1) **else** return(0).

Combined program:

$\pi_{a,b,c,d}$ (x,y): **if** (y=0) **then** $f_{a,b}(x)$ **else** $Ch_{c,d}$ (x).

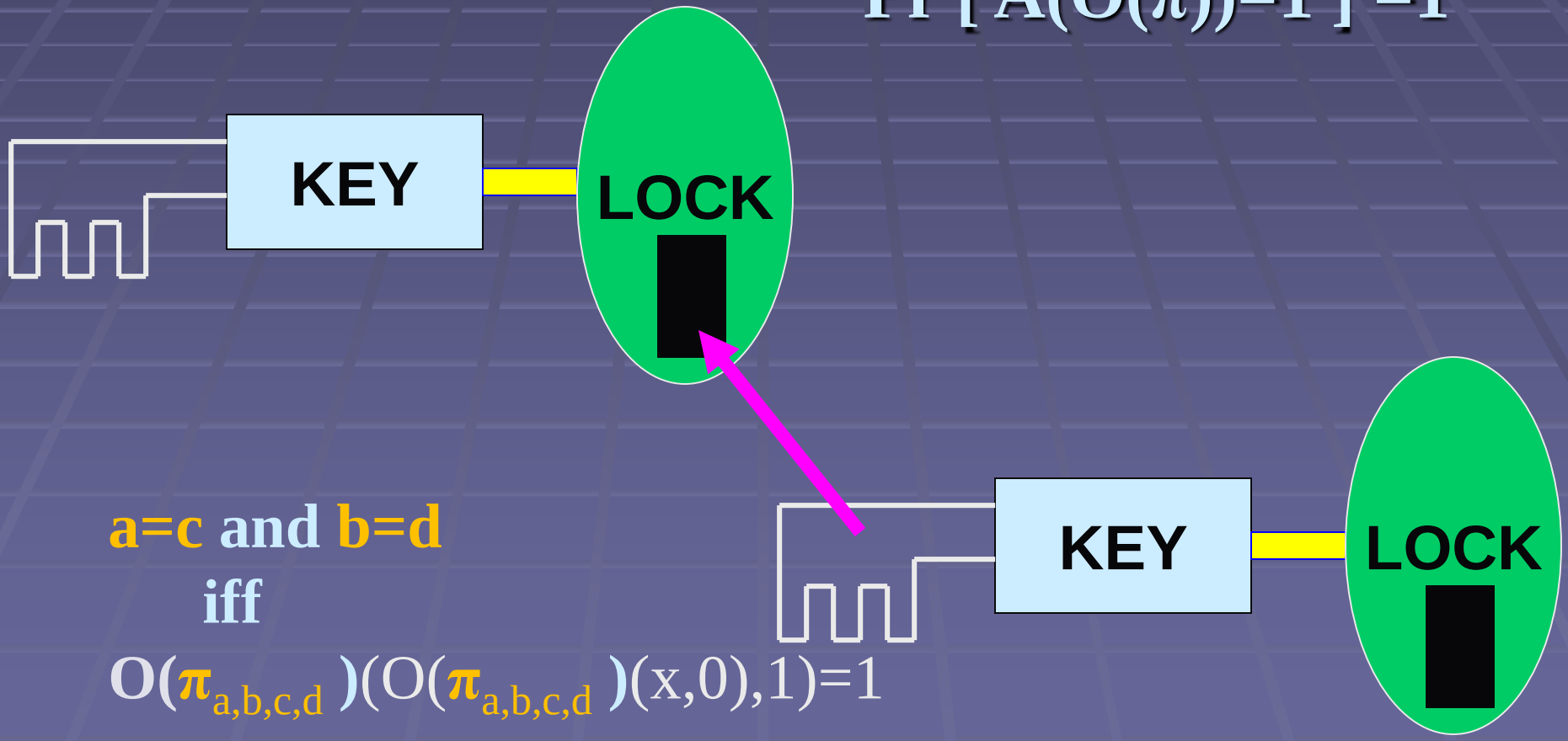
An adversary is aimed at learning, given $\pi_{a,b,c,d}$, whether a=c and b=d.

On the impossibility of secure obfuscation



On the impossibility of secure obfuscation

$\Pr [A(O(\pi))=1] = 1$



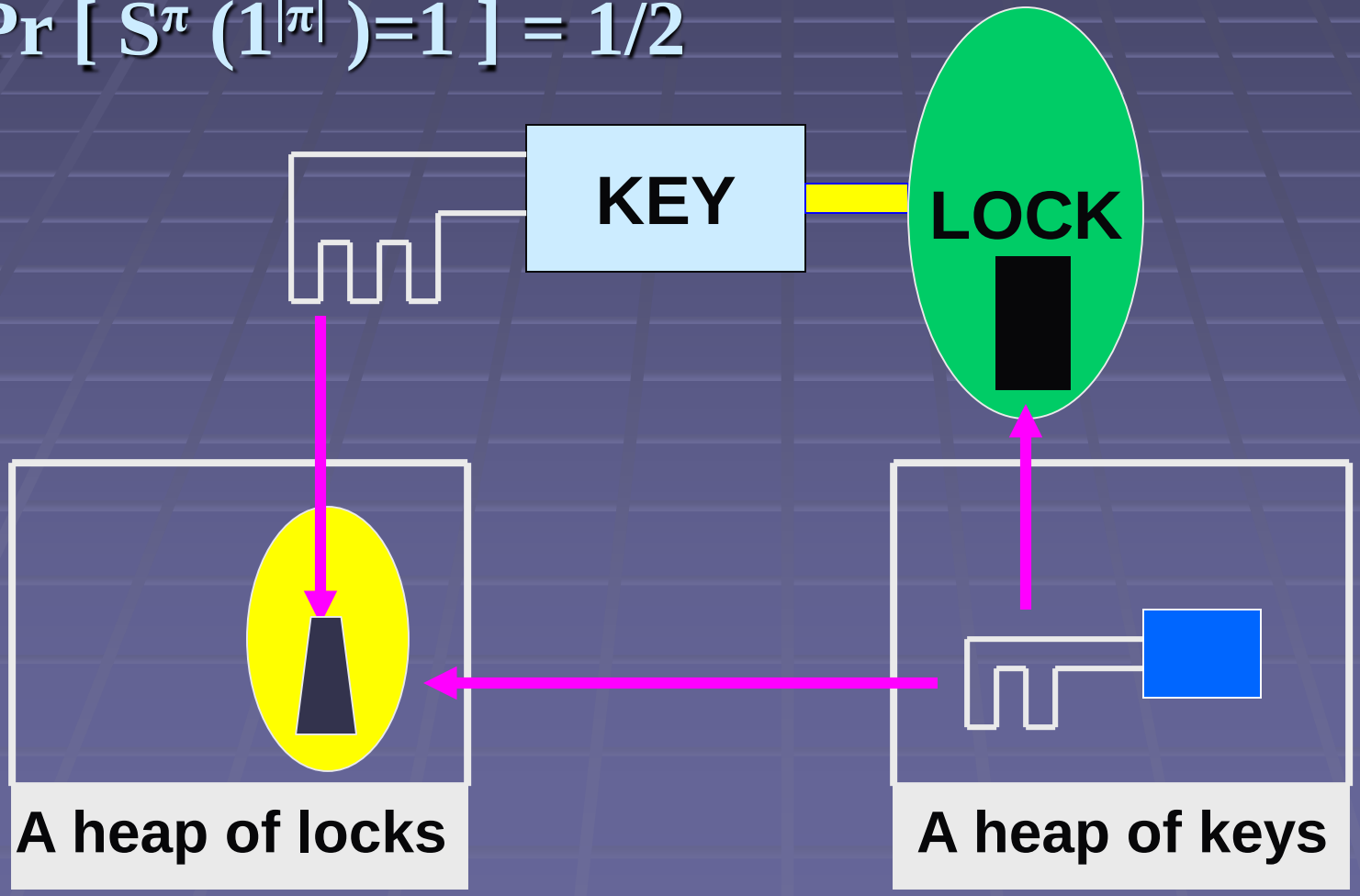
$a=c$ and $b=d$

iff

$O(\pi_{a,b,c,d})(O(\pi_{a,b,c,d})(x,0),1)=1$

On the impossibility of secure obfuscation

$\Pr [\, S^{\pi}(1^{|\pi|}) = 1 \,] = 1/2$



Obfuscation for software protection

An obfuscator O is a software protecting obfuscator if for every program π an obfuscated program $O(\pi)$ does not reveal more secrets than any other (arbitrary) program which computes the same function:

for every Probabilistic Polynomial Time Turing Machine (PPT) A there exists a PPT S and a negligibly small function neg(·) such that

$$\left|Pr\{A(O(M), N) = 1\} - Pr\{S^M(1^{\{|M|\}}), N\}\right| \leq neg(|M|)$$

for all programs M and N such that $M \sim N$ and $|N| = poly(|M|)$

Varnovsky N. P. **A note on the concept of obfuscation**, Tech. Reports of the Institute for system programming, 2004.

Obfuscation for software protection

Theorem. There exists a secure software protecting obfuscator for Deterministic Finite State Automata (DFA).

DFA obfuscator is but a DFA minimization algorithm.

This is a typical example of a trivial secure software protecting obfuscation by means of effective reduction of a program to the unique normal form (strong normalization).

Predicate obfuscation

An obfuscator O hides a predicate R if for every program π it is impossible to check efficiently whether an obfuscated program $O(\pi)$ has a property R :

"predicate hiding" property:

for every Probabilistic Polynomial Time Turing Machine (PPT) A there exists a PPT S and a negligibly small function $neg(\cdot)$ such that for all programs π

$$|Pr[A(O(\pi))=R(\pi)]-Pr[S^{\pi}(1^{|\pi|})=R(\pi)]| \leq neg(|\pi|),$$

Varnovsky N.P., Zakharov V.A. On the possibility of provably secure obfuscating programs. ``Perspectives of System Informatics'', 2004.

On the possibility of provably secure obfuscation of predicates

Theorem. If every program from a family ***F*** computes either zero function, or a point function, then secure obfuscation of the predicate

$$R(\pi) : \pi(x) \equiv 0$$

on the family ***F*** is possible under assumption that a one-way permutation exists.

Consider an ensemble $H=\{(S_n,D_n)\}$, where

$S_n=\{\pi_0\} \cup \{\pi^w_1 : w \text{ is a binary string of length } n\}$

The secret property R(π) is specified by the predicate

$\exists x\ (\pi(x)=1)$.

An adversary ***E*** is the set of all probabilistic polynomial time algorithms.

```
prog π0; \* no pass check
var  x : string, y : bit;
input (x) ;
y = 0; output (y);
end of prog
```

$D_n(\pi_0)=1/2$

```
prog π^w_1; \* actual pass check
var  x : string, y : bit;
const w : string;
input (x) ;
if x==w then y=1 else y=0;
output (y);
end of prog
```

$D_n(\pi^w_1)=1/2^{n+1}$

On the existence of relatively secure obfuscators

Alice

pass check π_1

π_2

π_3

pass check π_4

π_5

Which of her programs are protected?

Harry

Consider an arbitrary program π from S_n.

Define an obfuscator $O(\pi)=\Pi^{w,u,v,\sigma}$ as follows.

φ – a permutation on $\{0,1\}^n$;

if π is π_0 then choose randomly strings w,u and compute $v=\varphi(w)$,

$\sigma= \sum w_i u_i \bmod 2$;

if π is π^w_1 then choose randomly a string u and compute $v=\varphi(w)$,

$\sigma= 1+\sum w_i u_i \bmod 2$;

```
prog Π^{w,u,v,σ};
var  x : string, y : bit;
const u, v : string, σ: bit;
input (x) ;
if  φ(x) == v  then
     if  ∑x_i u_i == σ mod 2  then  y=0
                              else  y=1
else  y=0;
output (y);
end of prog
```

Topics for your essays

There are some other definitions of software obfuscation security.

Best possible obfuscation

Goldwasser S., Rothblum G.N. On best possible obfusction. Theory of Cryptography Conference 2007.

Indistinguishability obfuscation

Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vedhan S, Yang K., On the (Im)possibility of obfuscating programs. CRYPTO'01, 2001.

What are the relationships between these and other known definitions of obfuscation security?

Topics for your essays

Opaque predicates are very important for hampering formal analysis of programs. Not only Boolean formulae but some other structures can be used to build such predicates.

Usually every class of opaque predicates stems from some intractable mathematical problem.

Give 6 examples of hard problems in Boolean algebra, graph theory, combinatorics, arithmetic, etc. that can be used to build opaque predicates?

Topics for your essays

Program obfuscation is not only a nice theory but it is also a branch of software engineering. A lot of obfuscation tools have been implemented since 1997.

What is a current state of art in software obfuscation from the industrial point of view? What are the achievements nad drawbacks of modern obfuscation toolsets?

Thank you for attention!

Questions?