Applications of obfuscation to software and hardware systems

Victor P. Ivannikov

Institute for System Programming
Russian Academy of Sciences
(ISP RAS)
www.ispras.ru

Program obfuscation

is an efficient transformation **O** of a program **P** into an equivalent program **P'** such that **P'** is far less understandable than **P** (i.e. **P'** protects any secrets that may be built into and used by **P**).

A perfectly obfuscated program **P'** should complies with a “virtual black box” property: any information that can be extracted from the text of **P'** can be also extracted from the input-output behavior of **P'**.

Total obfuscation is a transformation of a program **P** into an equivalent program **P'** such that one can’t understand **what** **P'** is doing (i.e. one can’t learn the functionality of **P** from the text of **P’)**.

This guarantees a designer of the program **P** against its usage by an adversary in any applications as well as reverse engineering.

Weak obfuscation is a transformation of a program **P** into an equivalent program **P'** such that anyone even knowing the functionality of **P'** can't understand **how** **P'** operates. This means that one can't extract any useful information about data structures, algorithm, constants, etc. used in **P** from the text of **P'** and its input-output behavior specification.

In this case **P'** may be used in applications but any purposeful modification of its code (reverse engineering, inserting malicious fragments, breaking watermarks, etc.) is impossible.

The source paper:

W.Diffie, M.Hellman, New Directions in Cryptography (1976).

«A more practical approach to finding a pair of easily computed inverse algorithms E and D, such that D is hard to infer from E, makes use of the difficulty of analyzing programs in low level languages. Anyone who has tried to determine what operation is accomplished by someone else's machine language program knows that E itself (i.e. what E does) can be hard to infer from an algorithm for E. If the program were to be made purposefully confusing through the addition of unneed variables and statements, then determining an inverse algorithm could be made very difficult.

Of course, E must be complicated enough to prevent its identification from input-output pairs.

Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-way because it must be feasible to do the compilation, but infeasible to reverse the process.»

Cryptography and Obfuscation

Cryptography: C=E(M)

C is a ciphertext of a plaintext M .

Obfuscation: P'=O(P)

P' is a "ciphertext" of a source code P.

The main difference:

An obfuscated program ("ciphertext") P' has to be an executable program equivalent to P. Thus, obfuscation may be viewed as a semantic-preserving encryption of programs.

Applications of obfuscation

- To protect programs against reverse engineering and illegal modifications
- To protect software from illegal usage at the stage of distribution (with the help of watermarks)
- To provide security of mobile agents in the hostile environment
- To provide secret computation on the encrypted data (homomorphic encryption)
- To transform symmetric-key encryption algorithms into public-key ones

Theoretical Foundations of Program Obfuscation

The hardness of formal program analysis: undecidability of Halting Problem, Equivalence Problem, etc.

Theorem (Rice, 1953, Uspensky, 1954). Any non-trivial semanical property of computer programs is undecidable.

Obfuscation techniques

- C. Collberg, C. Thomborson, D. Low [1997] (taxonomy of obfuscating transformations)
- Wang C., Hill J., Knight J. Davidson J. [2000] (obfuscation as static analysis obstruction)
- Chow S., Gu Y., Johnson H., Zakharov V. [2001] (obfuscation via implantation of hard problems into a program)
- C. Linn, S. Debray [2003] (obfuscation via disrupting static disassembly process)
- X. Zhuang, R. Gupta [2003] (obfuscation with the help of program slicing techniques).

Formal definition of a perfect obfuscator
(B.Barak et al [2001])

A perfect obfuscator is a probabilistic algorithm O which satisfies the following three conditions:

(functionality): For every program π the string O(π) is a program that computes the same function as π.

(polynomial slowdown) : The size and running time of O(π) are at most polynomially larger than that of π.

(virtual black box): any probabilistic polynomial time (PPT) algorithm A (adversary), which has an access to the text of O(π) could achieve no better results than some PPT algorithm S which has oracle access to π (i.e. S uses π as a black box).

On the impossibility of “black-box” perfect obfuscators

Theorem. ”Virtual black box” perfect 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, LNCS, 2001, v.2139, p. 1-18.**

On the impossibility of "gray box" perfect obfuscators

An obfuscator O is called weakly perfect if for any program π an obfuscated program O(π) has virtual gray box property, i.e. any PPT A which has an access to the text of O(π) could achieve no better results than some PPT algorithm S which has access to the set Tr(π) of execution traces of program π.

Theorem. Weakly perfect obfuscators do not exist.

N.Varnovsky A note on the concept of obfuscation, Tech. Report of the Institute for System Programming, v. 6, 2004.

On the possibility of provably secure obfuscating programs

Theorem. **Password checking scheme can be obfuscated securely.**

N. Varnovsky, V. Zakharov, On the possibility of provably secure obfuscating programs.
2003, LNCS, v. 2890, p. 91-102 (ISP RAS)

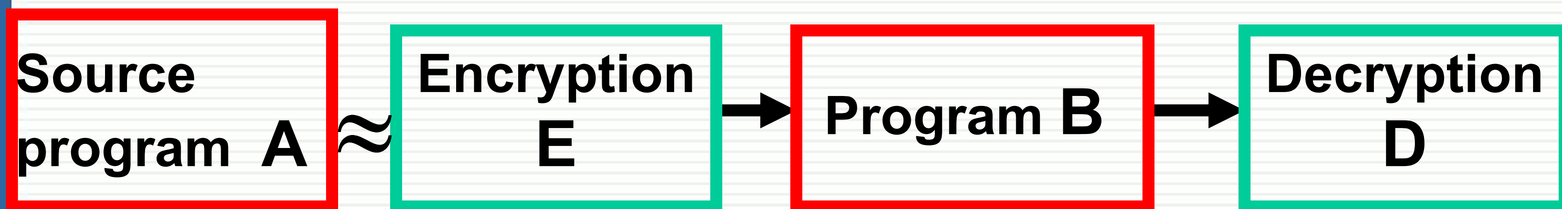
B. Lynn, M. Prabhakaran, A. Sahai, Positive results and techniques for obfuscation.
2004, LNCS, v. 3027, p. 20-39

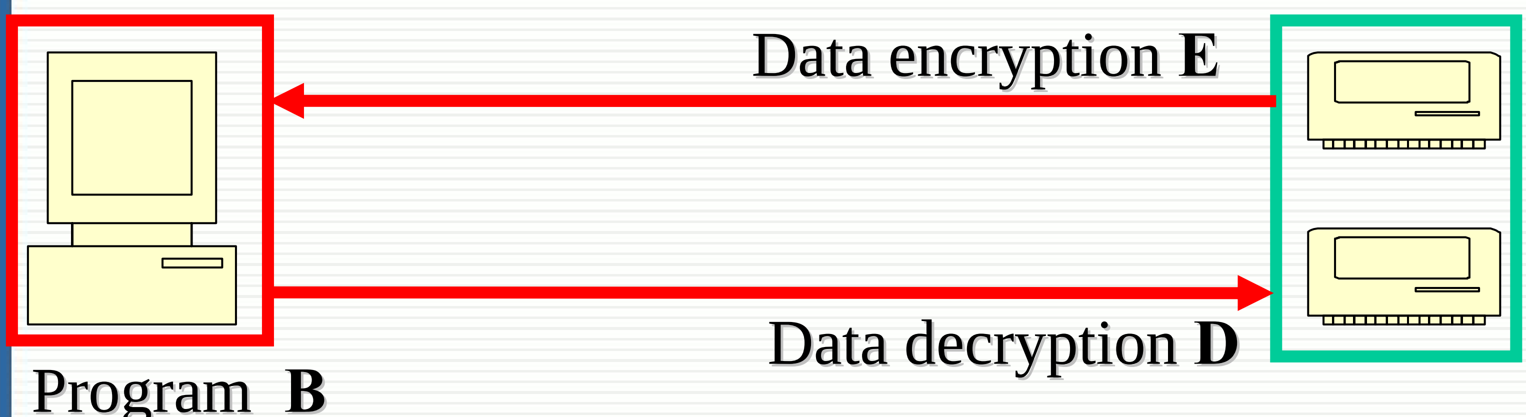
Computation on encrypted data

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**





Secure computations on encrypted data

Theorem. For any algebraic circuit **A** there exists an algebraic circuit **B** such that

1. Size(**B**) $\leq$ 2·Size(**A**)
2. there are efficient encoding **E** of inputs and decoding **D** of outputs such that

$$\mathbf{A} = \mathbf{D}(\mathbf{B}(\mathbf{E})) \qquad (*)$$

3. given a circuit **B** one can suggest exponentially many different circuits **A**, encodings **E** and decodings **D** satisfying ($*$).

A.V.Shokurov, An approach to quantitative analysis of resistance of data encodings in tamper-resistant software, Tech. Report of the Institute for System Programming v.6, 2004.

Directions for further theoretical research

1. Developing refined definitions of secure obfuscation (**obfuscation functionality, algorithms, constants**).

2. Open Problem:

   Is it possible to obfuscate securely finite state machines (automata)?

3. Challenge: to develop a formal concept of "semantic complexity" of programs (for measuring the complexity of program understanding).

   Analogy: Kolmogorov complexity

Implementations

- There are more than 20 Java byte-code obfuscators, including commercial. The most advanced – Zelix KlassMaster (www.zelix.com). It performs the following transformations:
  - Removal of debugging information.
  - Identifier renaming in the whole program.
  - Character string encoding.
  - Insertion of redundant JVM GOTO instructions to make reverse translation to Java source harder.

Taxonomy of Obfuscating Transformations

- Lexical obfuscation (comment removal, identifier renaming, structured construction removal, debugging info removal).
- Program control obfuscation.
- Program data obfuscation (string scrambling, array restructuring, etc).

C. Colberg at al. A Taxonomy of Obfuscating Transformations, 1997.

Program Control Obfuscation

- **Restructuring of the whole program:**
  - Function inlining,
  - Function outlining,
  - Function interleaving,
  - Function cloning,
  - Library calls elimination.
- **Transformation of a single function**:
  - «opaque» predicates, redundant code, «dead» code, use of identities,
  - Destructurization,
  - Basic block cloning,
  - Loop unrolling,
  - Loop fusion, loop fission, loop interleaving,
  - Creation of «dispatcher»,
  - Variable localization, variable globalization, variable reuse,
  - Increasing indirect level.

Example of lexical obfuscation

```
int fib(int n) {
  int a, b, c;
  a = 1;
  b = 1;
  if (n <= 1) return
  1;
  for (; n > 1; n--) {
    c = a + b;
    a = b;
    b = c;
  }
  return c;
  }
```

```
int f1(int r0) {
  int r1, r2, r3;
  r1 = 1;
  r2 = 1;
  if (r0 > 1) goto L22;
  return 1;
  L22: if (r0 <= 1)
  goto L23;
  r3 = (r1 + r2);
  r1 = r2;
  r2 = r3;
  r0--;
  goto L22;
  L23: return r3;
  }
```

How to measure the effect of transformation?

- Security of practical obfuscation techniques is in most cases cannot be proved.
- To estimate the effect of transformation Code Complexity metrics are used (Code Length, Cyclomatic Complexity, Fan In/Out Complexity, etc).
- **An adequate theoretical basis currently does not exist.**

Practical obfuscation study at ISP RAS

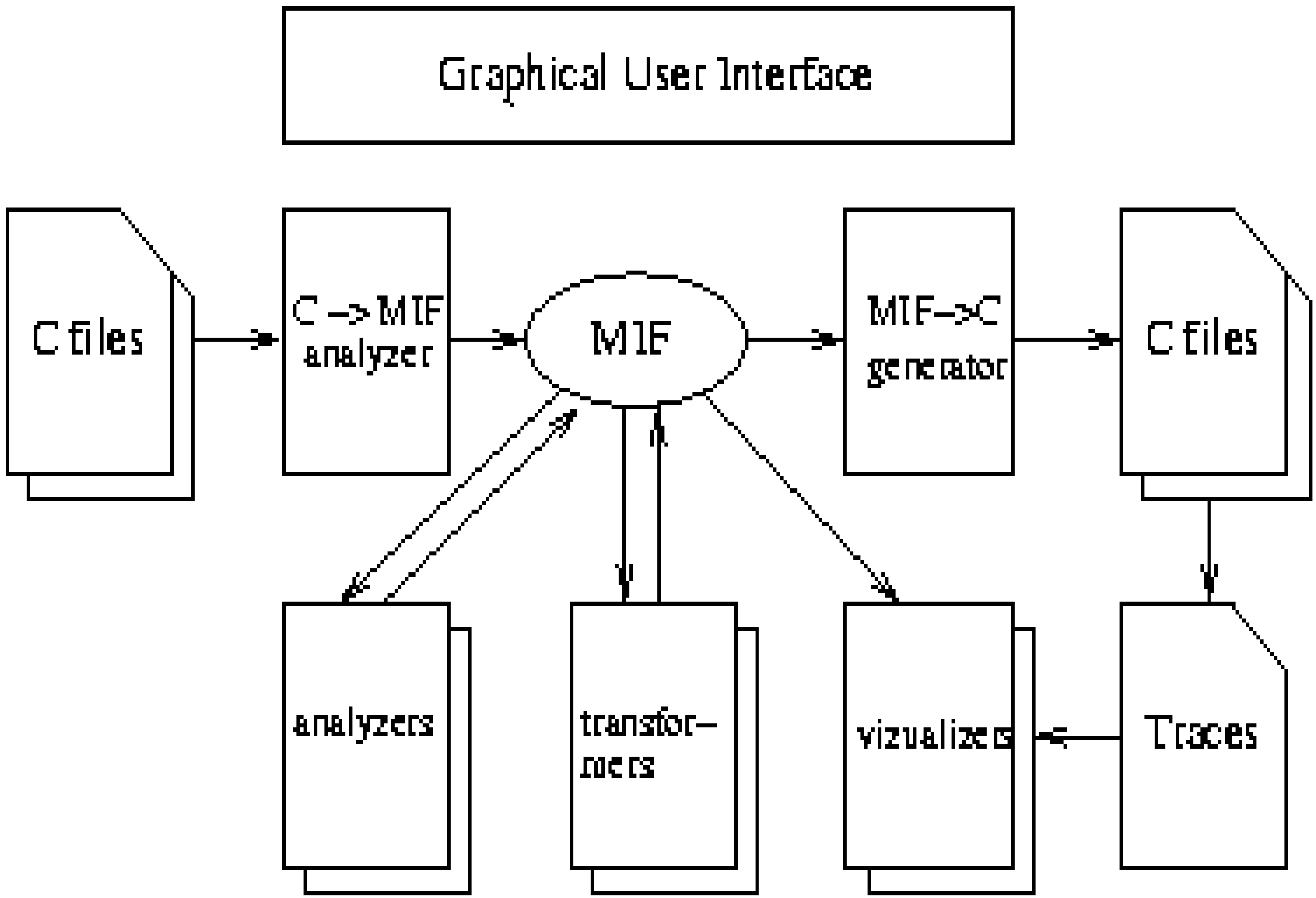
- The research is empiric, thus a programming environment to study analysis and transformation of programs is needed.

Requirements for an Integrated Research Environment (IRE)

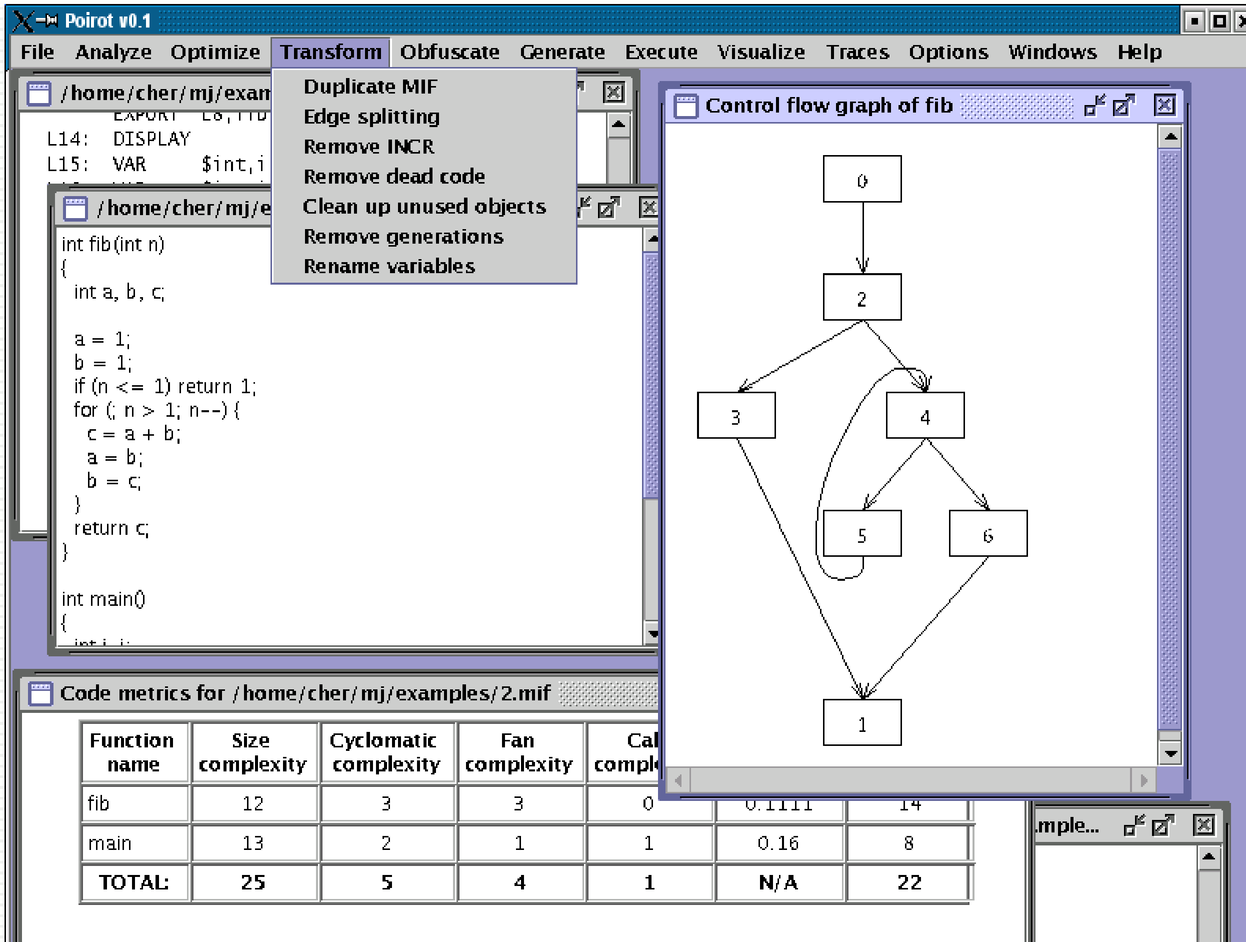
- Support for all the primary program analysys and optimization methods.
- Support for all primary program obfuscation methods.
- Open interface: possibility to add a new analysis or obfuscation method.

Uses of the IRE

- Research of program optimization methods (parallelizing, profile-based optimizations, etc).
- Study of program obfuscation and analysis of obfuscated programs.
- As a tool for detection of security vulnerabilities and malicious code (in perspective).


Graphical User Interface
C files
C -> MIF analyzer
MIF
MIF->C generator
C files
analyzers
transfor-mers
vizualizers
Traces

IRE User Interface



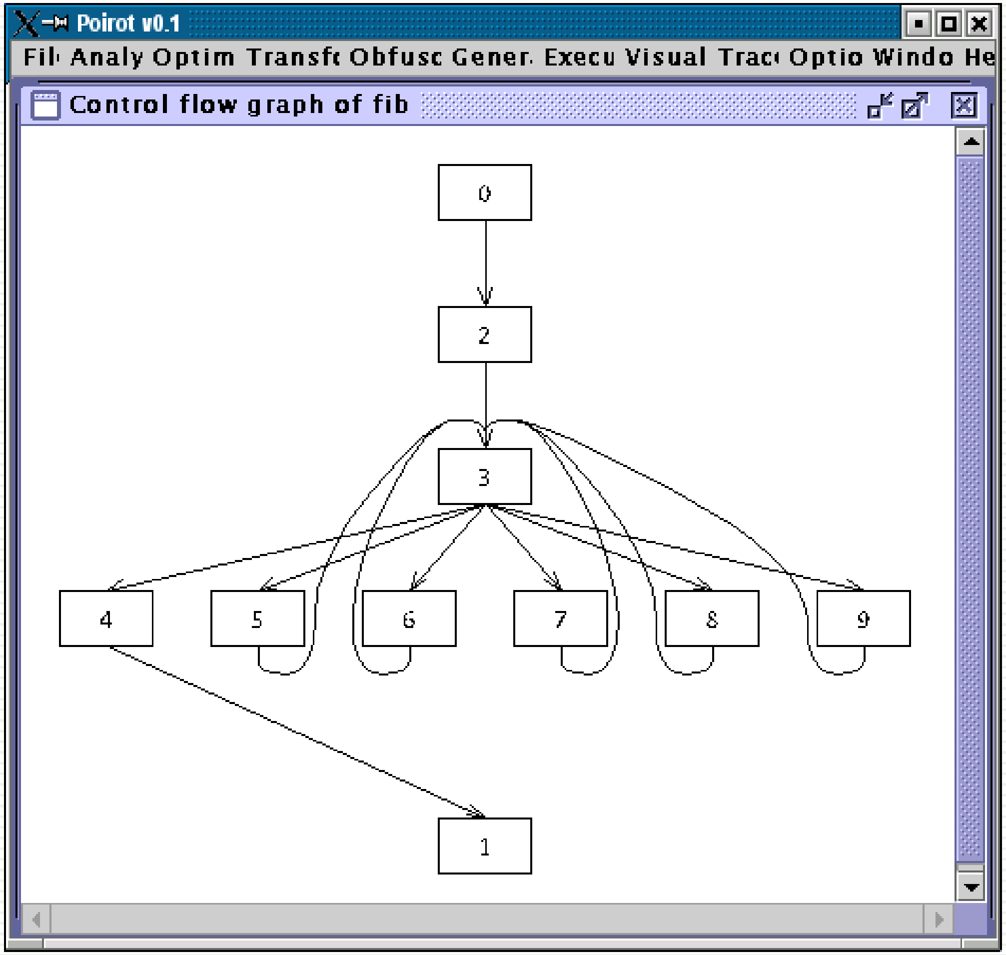
Analyzing/optimization tools

- Inter/intra-procedural alias and range analysis.
- Program dependency analysis.
- Constant/copy propagation, dead code elimination, common subexpression elimination, invariant code motion.
- Basic block node/edge profiling, tracing, control-flow graph recovery.

Obfuscation tools

- Identifier renaming, structure elimination.
- Dispatcher construction.
- Static and tracing-resistant obfuscation based on control-flow transformation (PhD thesis, 2003, A. Chernov).
- Obfuscation using cryptography primitives (MSc thesis, ISP RAS 2004, A. Lokhmotov).

Dispatcher transformation



Dispatcher table encoding

- A method is developed, which uses one-way functions and pseudo-random generators.
- The method has provable cryptographic resistance against program static analysis.
- One-way function and pseudorandom functions are based on knapsack problem:

$$f_{\boldsymbol{a}}(x) = \sum_i x_i a_i \mod 2^m,$$

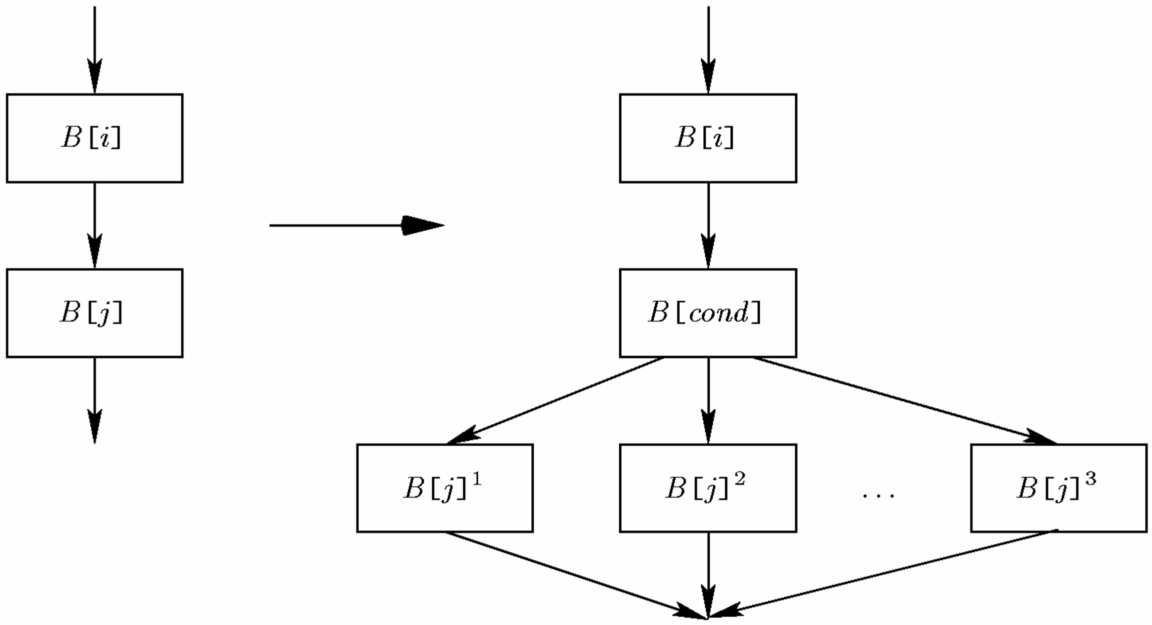
$x = (x_1 x_2 \ldots x_n)$, $x_i \in \{0,1\}$; $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$, $a_i \in \{0,1\}^m$

Resistance against profile-based analysis

- Increase the complexity of the CFG by adding new edges. The added edges are not “dead”.
- Increase the data complexity by generating dummy code and its environment.
- Stop data-flow analysis methods by adding “false” data dependencies between the dummy and the original code.

A. Chernov. A New Program Obfuscation Method. In *Proceedings of the Adrei Ershov Fifth International Conference “Perspectives of Systems Informatics”. International Workshop on Program Understanding*, Novosibirsk, July 14-16, 2003.

Basic block cloning



Obfuscating functions in hardware

- The goal of hardware obfuscation is to protect secret information in circuit design.
- Financial/military applications.