DC Sec

Leibniz Universität Hannover

Homomorphic Cryptography for Cloud Computing

4th DAAD Summer School on Current Trends in Distributed Systems 2012

Henning Perl

Friday, September 7th 2012

Outline

Real World Applications

What is Homomorphic Cryptography?

Homomorphism := **Structure-preserving map**

w.r.t. operations:

$$\varphi(a) + \varphi(b) = \varphi(a + b)$$

$$\varphi(a) \cdot \varphi(b) = \varphi(a \cdot b)$$

Crypto scheme:

$$\varphi \equiv \text{Encrypt}$$

$$\varphi^{-1} \equiv \text{Decrypt}$$

Conclusions:

- Evaluation of arbitrary formulas with $+$ and $\cdot$
- Decryption yields sum or product

$\Rightarrow$ **We can do stuff with ciphertexts!**

What All Can We Do?

$\Rightarrow$ **We can do stuff with ciphertexts!**

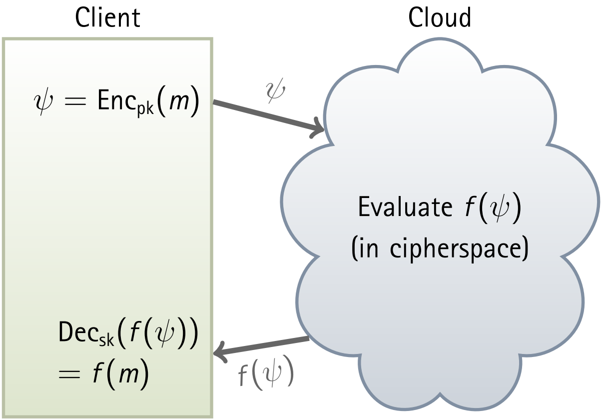
Assume a homomorphic encryption scheme. What do we get?

- Homomorphism: Addition/Multiplication of ciphertexts yields sum/product after decryption
- Next step: Plaintexts are Bits ($\mathcal{P} = \mathbb{Z}/2\mathbb{Z}$)
- Then:

$$\left.\begin{array}{llll} a+b & \bmod 2 \rightsquigarrow a \oplus b & \text{(xor)} \\ a \cdot b & \bmod 2 \rightsquigarrow a \wedge b & \text{(and)} \end{array}\right\} \rightsquigarrow \text{Arbitrary boolean circuits}$$

- Finally: Build a CPU out of boolean circuits
  $\rightsquigarrow$ Arbitrary programs in cipherspace.

Use-case in Cloud Computing



Notation:

m	plaintext
ψ	ciphertext
f	program
pk	public key
sk	secret key
Enc_{pk}	encryption
Dec_{sk}	decryption

History of Homomorphic Cryptography

"Holy Grail" of cryptography for a long time

1978 Posed as open problem by Rivest *et al.*

2005 Evaluate 2-DNF formulas on ciphertexts, Boneh *et al.*

2009 Fully homomorphic encryption using ideal lattices by Craig Gentry

2010 Fully homomorphic encryption over the integers by van Dijk *et al.*

2012 Fully homomorphic encryption without bootstrapping by Brakerski *et al.*

Gentry's original scheme

Common construction:

- Plaintext, ciphertext are rings (operations $+$ and $\cdot$)
- Encryption is a homomorphism from plaintext to ciphertext
- Operations on ciphertexts add noise
- Decryption succeeds as long as noise remains within bounds

"Cleaning" the ciphertext (*bootstrapping*):

- Represent decryption function as boolean circuit
- Decrypt a ciphertext in ciphertext space $\rightsquigarrow$ "cleaner" new ciphertext
- Requirements:
  - Decryption circuit must be shallow enough
  - Called *bootstrappable*-property

A simple homomorphic scheme by example

A simple homomorphic scheme by example

Goal:

- Simple, easy to understand homomorphic scheme
- Symmetric scheme (with key p)
- Hardness based on prime number factorization
- Not bootstrappable (i.e. not *fully* homomorphic)

Notation:

- $\mathbb{P}$ – number of primes
- $a \overset{R}{\leftarrow} A$ – choose a from set A with uniform distribution

Keygen, Encrypt, Decrypt

$p \leftarrow \text{Keygen}(\lambda)$

1: **return** random λ-bit prime

$c \leftarrow \text{Encrypt}_\lambda(m, p)$

1: $r \xleftarrow{R} \mathbb{N}$
2: $q \xleftarrow{R}$ random λ-bit number
3: **return** $m + 2r + pq$

$m \leftarrow \text{Decrypt}(c, p)$

1: **return** $(c \mod p) \mod 2$

Correctness:

$$\begin{aligned} & \text{Dec}(\text{Enc}(m, p)) \stackrel{!}{=} m \\ \Leftrightarrow\ & ((m + 2r + pq) \mod p) \mod 2 \\ =\ & (m + 2r) \mod 2 = m \end{aligned}$$

Bit Operations

Xor(c_1, c_2)

1: **return** $c_1 + c_2$

And(c_1, c_2)

1: **return** $c_1 \cdot c_2$

Correctness

Remember: $\text{Encrypt}(m, p) := c \leftarrow m + 2r + pq$

Xor / Addition

$$\text{Decrypt}(\text{Xor}(c_1, c_2), p) = \text{Decrypt}(m_1 + 2r_1 + pq_1 + m_2 + 2r_2 + pq_2, p) =$$
$$\left(((m_1 + m_2) + 2(r_1 + r_2) + p(q_1 + q_2)) \mod p\right) \mod 2 = m_1 + m_2$$

And / Multiplication

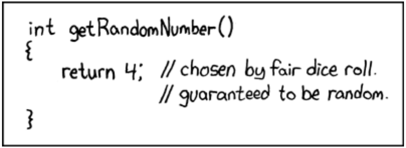
$$\text{Decrypt}(\text{And}(c_1, c_2), p) =$$
$$\text{Decrypt}((m_1 + 2r_1 + pq_1)(m_2 + 2r_2 + pq_2), p) =$$
$$\left((m_1 \cdot m_2 + 2r_2 m_1 + pq_2 m_1 + 2r_1 c_2 + pq_1 c_2) \mod p\right) \mod 2 = m_1 \cdot m_2$$

Putting in Numbers

- $p \leftarrow \text{Keygen}()$: Choose $p = 23$
- $c_0 \leftarrow \text{Encrypt}(0, p)$:
  - Choose $q \leftarrow 5$
  - Choose $r \leftarrow 3$
  - Choose

    $c_0 \leftarrow 0 + 2 \cdot 3 + 5 \cdot 23 = 121$
- $c_1 \leftarrow \text{Encrypt}(1, p)$:
  - Choose $q \leftarrow 4$
  - Choose $r \leftarrow 4$
  - Choose

    $c_1 \leftarrow 1 + 4 \cdot 3 + 4 \cdot 23 = 101$



Examples

$c_0 \oplus c_1 = 121 + 101 = 222$

$222 \mod 23 = 15 \rightsquigarrow 1$

$c_0 \wedge c_1 = 121 \cdot 101 = 12221$

$12221 \mod 23 = 8 \rightsquigarrow 0$

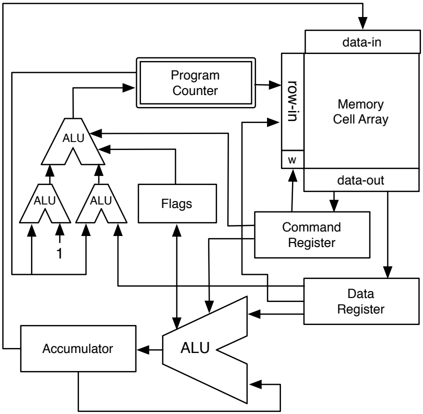
The Problem With The Noise

Remember: $m \leftarrow \text{Decrypt}(c) = (c \mod p) \mod 2$

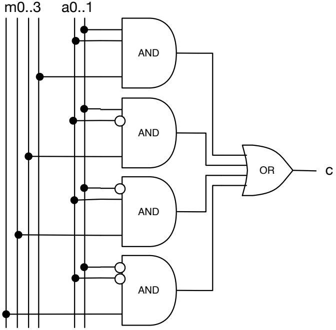
Decryption only works iff. $a + b < p$, $a \cdot b < p$

	$p \cdot q_1$	$2 \cdot a_1$	m_1
$\oplus$	$p \cdot q_2$	$2 \cdot a_2$	m_2
$=$	$p \cdot (q_1 + q_2)$	$2 \cdot (a_1 + a_2)$	$m_1 + m_2$

Encrypted CPU — Overview



Encrypted memory access — reading



m single-bit memory cells

a address

c selected cell

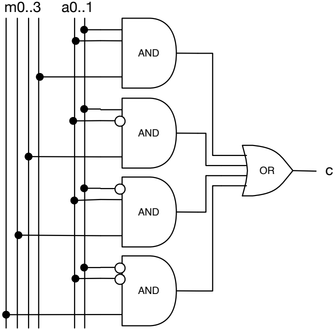
Selection circuit

$$c(a, m) =$$
$$(\neg a_0 \wedge \neg a_1 \wedge m_0) \oplus (a_0 \wedge \neg a_1 \wedge m_1) \oplus$$
$$(\neg a_0 \wedge a_1 \wedge m_2) \oplus (a_0 \wedge a_1 \wedge m_3)$$

Analysis

- Two memory addresses indistinguishable
- Access pattern hidden
- $\Rightarrow$ *Oblivious read access*

Encrypted memory access — writing



m single-bit memory cells

a address

c selected cell

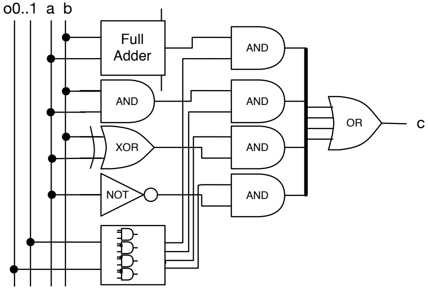
Selection circuit

$$c(a, m) =$$
$$(\neg a_0 \wedge \neg a_1 \wedge m_0) \oplus (a_0 \wedge \neg a_1 \wedge m_1) \oplus$$
$$(\neg a_0 \wedge a_1 \wedge m_2) \oplus (a_0 \wedge a_1 \wedge m_3)$$

Write access (write d to address a)

- For each memory cell m:
  - $m \leftarrow (c(a, m) \wedge d) \oplus (\neg c(a, m) \oplus m)$
- $\Rightarrow$ *Oblivious write access*

Encrypted Arithmetic–Logical Unit



o opcode

a first parameter

b second parameter

Opcodes

o_0	o_1	Output	o_0	o_1	Output
0	0	$a + b$	1	0	$a \oplus b$
0	1	$a \wedge b$	1	1	$\neg a$

- ALU function selection same as memory selection
- From here:
  - Fix machine word (i.e. 8 bits)
  - Add circuits for full adder, carry-flag, zero-flag etc.

Plugging it together

- ALU for arithmetic and logic operations
- Group of smaller ALUs for program flow control etc.
- Registers: encrypted bit columns
- Memory: memory cells (encrypted bit columns) with access logic
- Simple processor cycle:

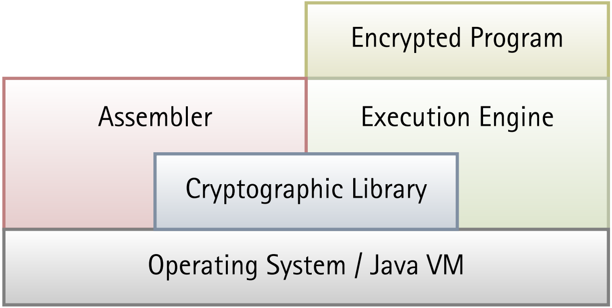
  FETCH1 read memory cell pointed at by program counter

  FETCH2 read memory cell pointed at by fetched operand

  EXEC execute operation in command register

  WRITE write results to memory
- *Note:* Every cycle performs all three memory access operations (needed for obliviousness)

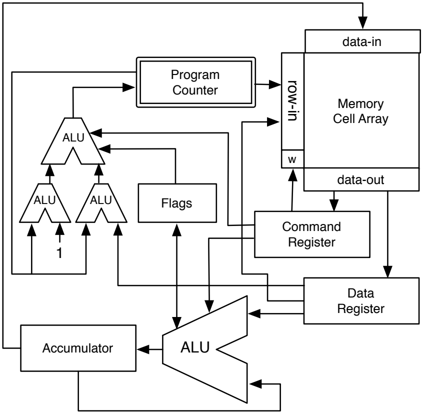
System architecture



Implementation Details

- Memory word length: 13 bits = 8 bits data + 5 bits opcode
- Architecture independent from concrete cryptosystem
- One cycle takes $\approx$ 2 ms (2.4 GHz Intel Core 2 Duo)

Encrypted CPU — Summary



- Data is encrypted
  - Read accessible
  - Write accessible
- Program is encrypted
  - All opcodes are encrypted
  - ALU output is encrypted

Code at http://hcrypt.com/shape-cpu/

Encrypted, Turing–complete machine

Real World Applications

1 Introduction

2 A simple homomorphic scheme by example

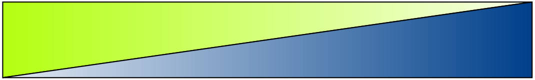
3 An Encrypted CPU for Homomorphic Cryptography
- Encrypted memory access – reading
- Encrypted memory access – writing
- Encrypted Arithmetic-Logical Unit

4 Real World Applications
- Why are we not done yet?
- Searching on Encrypted Data

5 Conclusion

Why are we not done yet?

Performance



Confidentiality

unencrypted hybrid algorithm encrypted CPU

Unencrypted: very fast, familiar tools (compiler etc.)

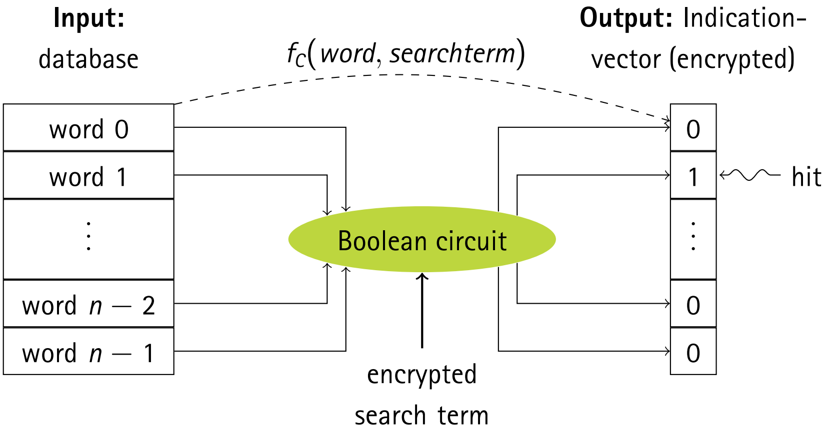
but requires trust

Encrypted CPU: Turing-complete, encrypts data and program

but bad performance

Hybrid algorithm: seeks a compromise: protects confidential parts (just data, just part of the data etc.), performs better than encrypted CPU

Search with Boolean circuits



Exact Search

Boolean circuit f_c: Conjunction over single letters

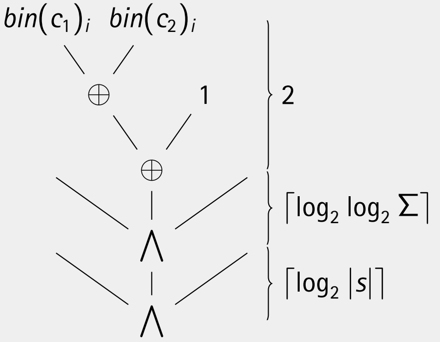
$$\bigwedge_{i=0}^{|s|-1} (w_i =_c s_i)$$

Comparison $=_c$: Conjunction over binary representation

$$\bigwedge_{i=0}^{\lceil \log_2 \Sigma \rceil - 1} bin(c_1)_i \oplus bin(c_2)_i \oplus 1$$

$a \oplus b \oplus 1$	0	1
0	1	0
1	0	1

Depth of f_c:



Use Case: Search on Human Genomes

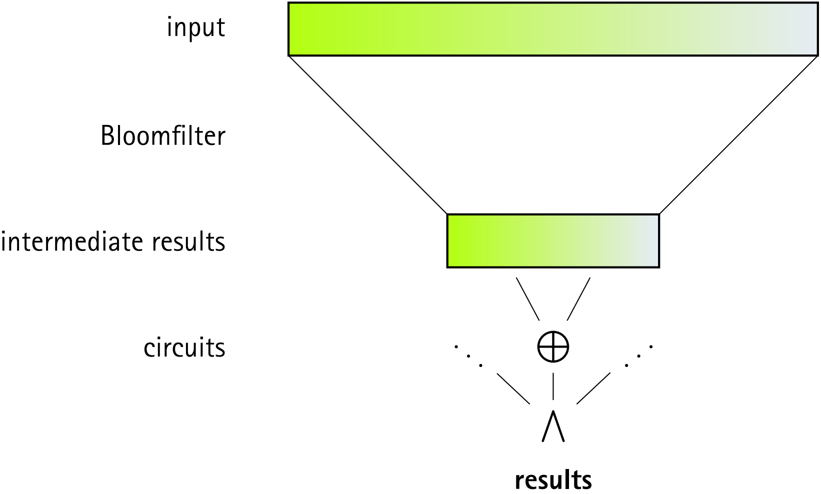
Motivation:

- Large database of human genomes
- Future: personalized medications and therapies
- Depend on patients' DNA
- $\Rightarrow$ database is public, query is confidential

But ...

- too much data (100 MB up to several GB)
- kills performance even with customized circuit
- Solution: Prefilter results $\rightsquigarrow$ Bloomfilter Search
- $\rightsquigarrow$ Hybrid algorithms

Hybrid Algorithm for Searching



Bloomfilters for words and sets

n hash functions $f_0 \dots f_{n-1}$; Bloomfilter length m; alphabet Σ; word $w \in \Sigma$

helper function $b(f(w)) = (\dots, 0, \underbrace{1}_{\text{at } f(w)}, 0, \dots) \in [0,1]^m$

Definition of $\mathcal{B}(w)$:

$$\mathcal{B} : \begin{array}{l} \Sigma^* \to [0,n]^m \\ w \mapsto \sum_{i=1}^{n} b(f_i(w)) \end{array}$$

$f_1(w) = 2$ $\quad$ $f_0(w) = 7$

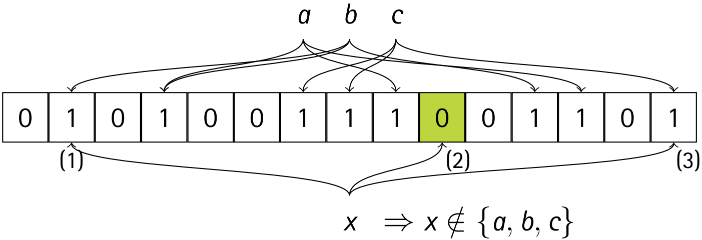
0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	0	1	0

Definition of $\widehat{\mathcal{B}}(M)$:

$$\widehat{\mathcal{B}} : \begin{array}{l} \wp(\Sigma^*) \to [0,n]^m \\ M \mapsto \sum_{w \in M} \mathcal{B}(w) \end{array}$$

	0	1	2	3	4	5	6	7	8
w_0	0	0	1	0	0	0	0	1	0
w_1	1	0	0	0	0	1	0	0	0
$\{w_0, w_1\}$	1	0	1	0	0	1	0	1	0

Set membership with Bloomfilters



Bloomfilter with three hash functions

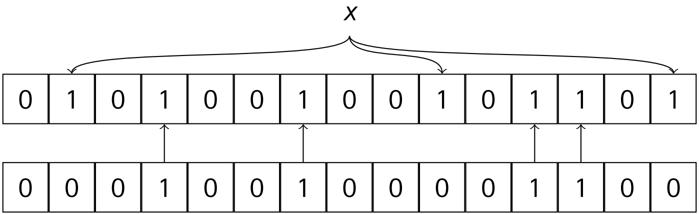
$$\mathcal{B}(w) \notin \widehat{\mathcal{B}}(M) :\Leftrightarrow \exists i \in \text{index set} : \mathcal{B}(w)[i] > \widehat{\mathcal{B}}(M)[i] \Rightarrow w \notin M$$

$$\mathcal{B}(w) \in \widehat{\mathcal{B}}(M) :\Leftrightarrow \forall i \in \text{index set} : \mathcal{B}(w)[i] \leq \widehat{\mathcal{B}}(M)[i] \Rightarrow w \in M$$

Obfuscation of Bloomfilters

Goal: Fuzzyness in set membership: make $w \in M$ more probable

Method: Obfuscation with parameter λ (here $\lambda = 4$)

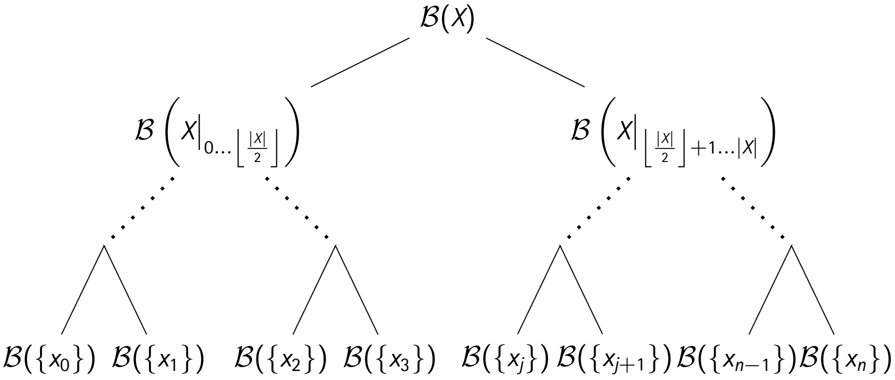


Result: Given a Bloomfilter $\mathbf{B} = \mathcal{B}^{k,m}(A)$ for a set A and a (plain) Bloom filter $\mathbf{b} = \mathcal{B}^{k,m}(a)$ as well as an obfuscated version $\mathbf{b}' = \text{obfuscate}(\mathbf{b}, m, \lambda)$:

- $\mathbf{b} \in \mathbf{B} \Rightarrow \mathbf{b}' \in \mathbf{B}$
- $\mathbf{b}' \in \mathbf{B} \Rightarrow \Pr[\mathbf{b} \in \mathbf{B}] = 1/\binom{k+\lambda}{k} \rightsquigarrow$ hiding in x %

Index search with Bloomfilters

- Bloomfilter tree: Index for database X
- Search: divide-and-conquer in $\mathcal{O}(\log(X))$



Conclusion

- Homomorphic cryptography is a powerful tool
- Relatively new development, more to come
- Promising new applications, especially with security / performance tradeoffs
- Code available at `http://hcrypt.com`

Further research

- Faster homomorphic cryptoschemes
- More hybrid algorithms $\rightsquigarrow$ more applications

Thank You!