CSE/CEN 598
Hardware Security & Trust

Secure Computation Approaches: Homomorphic Encryption

Prof. Michel A. Kinsy

Secure Computation Approaches

Multi-Party Computation (MPC)

Pros
- Low compute requirements
- Easy to accelerate
- Provably secure
- Supports multiple threat models
- Easy to map existing algorithms

Cons
- High communication costs
- High latency
- Information theoretic proofs are weaker than PKE ones

Fully Homomorphic Encryption (FHE)

Pros
- Very low communication costs
- Requires a single round of communications, i.e., "fire and forget"
- Useful when one side is limited in compute / memory / storage
- Provably secure – relies on strength of PKE

Cons
- Very high computational requirements
- Harder to accelerate
- Mapping existing algorithms to FHE may be difficult

Trusted Execution Environments (TEE)

Pros
- No communication required
- Trivial to accelerate
- Great support for existing software

Cons
- Weaker security guarantees
- Cannot stop determined adversaries
- Historically plagued by vulnerabilities and breaches
- Long term deployment is difficult – TEE's can 'run out' of entropy / CRP's, etc.

Outsourced Computation

- Cloud storage and computing have many advantages
  - The rise of connected and sensor-based devices have led to cloud computing being used as a commodity technology service

Google Cloud

Microsoft Azure

Alibaba Cloud

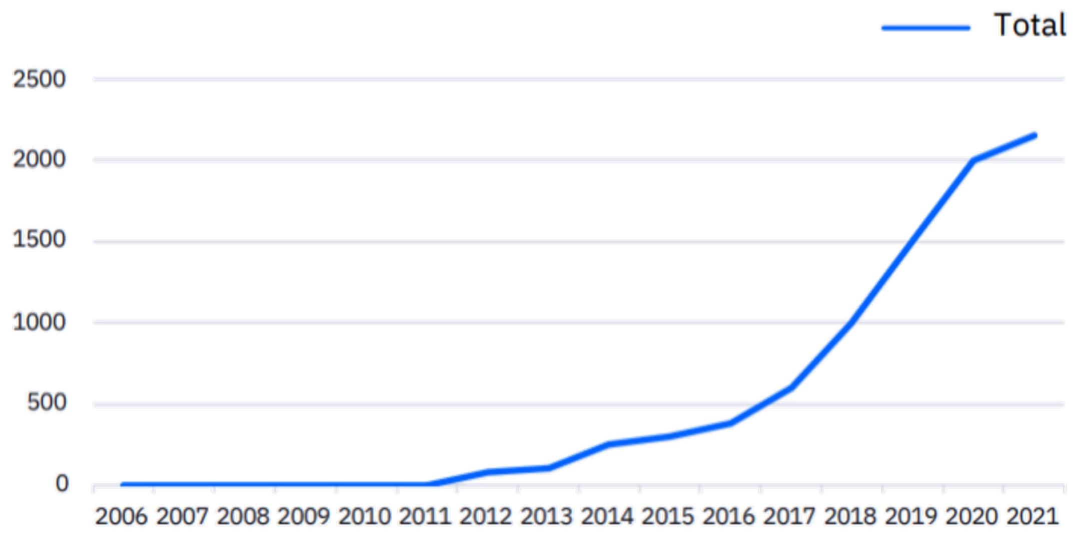
IBM Cloud

aws

SECURE?

Outsourced Computation

- One of the key issues with cloud-based computation is data privacy
  - Sensitive data is stored and computed over the cloud, which at most times, is a shared resource
  - We currently have more than 2,500 cloud vulnerabilities
  - 150% increase just in the last five years



Number of cloud vulnerabilities tracked by IBM Security X-Force

Outsourced Computation

2017 — Verizon

Data of at least 6 million users are exposed due to "human error" from misconfigured Amazon S3 cloud Server.

2019 — Capital One

Former worker of AWS illegally accessed into Capital One's AWS cloud server and leaked personal data of 106 million people.

2019 — Best Western

179 GB database included personal data of members of the U.S government, military, and DHS are exposed via unsecured Elastic-search database hosted on AWS.

2020 — Alibaba

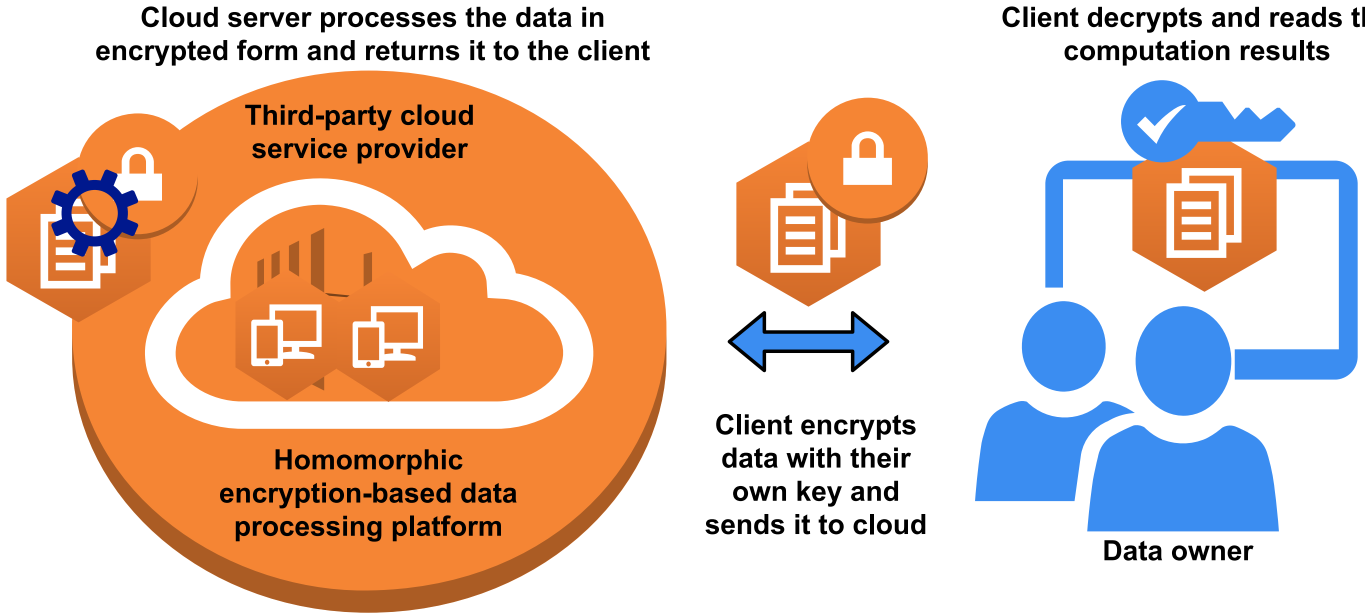
Alibaba Cloud's staffs obtained client contact information and leaked it to a third-party partner without consent.

2021 — Microsoft

A flaw in Microsoft's Azure Cosmos DB database left more than 3,300 Azure customers (including many Fortune 500 companies) open to complete unrestricted access by attackers.

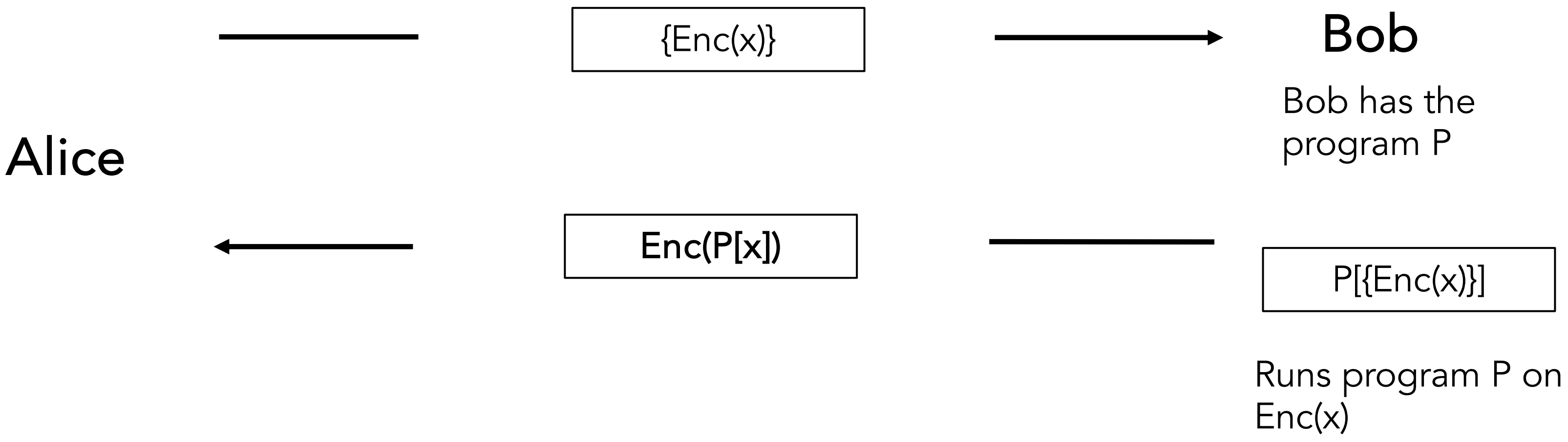
Homomorphic Encryption

- What is Homomorphic Encryption (HE)?
  - Encryption scheme that allow computation on encrypted data without decryption
  - Homomorphic encryption can be used along with cloud services to perform computations on encrypted data, guaranteeing data privacy



Homomorphic Computation

- Homomorphic computation is a form of of computation that allows computation on encrypted data
  - The result of this encrypted data processing itself is encrypted
  - But when the result is decrypted, it should match the output of the program if it runs directly on the encrypted input data



Overview Homomorphic Encryption (HE)

- An encryption scheme is called homomorphic over an operation '*' if it supports the following
  - $\forall (m_1, m_2) \in M, Enc(m_1) * Enc(m_2) = Enc(m_1 * m_2)$
  - Where Enc is the encryption algorithm and M is the set of all possible messages
- Supporting addition and multiplication operations is sufficient to create an encryption scheme that the homomorphic evaluation of an arbitrary function
  - Any Boolean circuit can be represented using only XOR (addition) and AND (multiplication) gates

Homomorphic Encryption

- Homomorphic Encryption
  - Is a form of encryption that allows computations to be carried out on ciphertext
  - Generates an encrypted result
  - The result when decrypted matches the result of operations performed on the plaintext
- Formally,
  - $Eval_E(f, c_0, c_1, \ldots, c_n)$
- Example:
  - Enc(key, 2) = $, Enc(key, 3) = %
  - Eval(+, $, %) = #
  - Dec(key, #) = 5

Simple Illustrative Example

- Function to compute
  - f is a simple addition
    - Such that $y = f(x_1, x_2, ..., x_k)$
  - $y = x_1 + x_2 + ... + x_n = \sum_{i=1}^{n} x_i$
    - n is number of terms
- We will define our encryption function as
  - $Enc\ (x_i) = (x_i + p)*q$ where p and q determine the key $k(p,q)$ and it is private
- We define the decryption function as
  - $Dec\ (Y, n) = Y/q - n*p$

Simple Illustrative Example

- Homomorphically compute
  - $f_{HE}(5, 9, 3)$
- Computation Process
  - Key Generation
    - Pick p and q
    - k(7, 2)
  - Encrypt the terms
    - Enc(5) = (5+7) * 2 = 24
    - Enc(9) = (9+7) * 2 = 32
    - Enc(3) = (3+7) * 2 = 20
  - Evaluation
    - f(24, 32, 20) = 24 + 32 + 20 = 76
  - Decrypt the result
    - Dec(76, 3) = (76/2) + (3*7) = 17

Homomorphic Computation

- Fully Homomorphic Encryption (FHE)
  - It was first defined in 1978 under privacy homomorphism
    - For the purpose of searching encrypted data
  - Various approaches
    - Multiplicatively homomorphic by RSA and El Gamal Additively homomorphic by GM and Paillier
    - Quadratic formulas by BGN'05 and GHV'10a
  - Recent major advances
    - First Construction of fully homomorphic encryption by Gentry [2009]
      - Using algebraic number theory - ideal lattices

Overview Homomorphic Encryption (HE)

- Techniques to compute on encrypted data can be classified in three (3) categories
  - Partially Homomorphic Encryption (PHE)
    - Allowing only one type of operation with an unlimited number of times
  - Somewhat Homomorphic Encryption (SHE)
    - Allowing some types of operations with a limited number of times
  - Fully Homomorphic Encryption (FHE)
    - Allowing an unlimited number of operations with unlimited number of times

Overview Homomorphic Encryption (HE)

- Techniques to compute on encrypted data can be classified in three (3) categories
  - Partially Homomorphic Encryption (PHE)
    - Unlimited add OR multiplication
  - Somewhat Homomorphic Encryption (SHE)
    - Limited addition AND multiplication
  - Fully Homomorphic Encryption (FHE) $\Leftarrow$ SHE + Bootstrapping
    - Unlimited addition AND multiplication

Homomorphic Encryption Approaches

- Popular Homomorphic Encryption Schemes
  - TFHE – Fast Fully Homomorphic Encryption - 2016
    - Support homomorphic evaluation on logic gates (AND, OR, NAND, NOT, MUX, etc.)
    - Best for operation on individual bits
  - BGV (Brakerski-Gentry-Vaikuntanathan - 2011) and BFV (Brakerski/Fan-Vercauteren - 2012)
    - Exact arithmetic on vectors of numbers
    - Best for vectorized operation over finite fields
  - CKKS (Cheon, Kim, Kim and Song – 2016)
    - Approximate arithmetic on vectors of numbers
    - Best for vectorized operation over real numbers

Overview Homomorphic Encryption (HE)

- A Homomorphic Encryption algorithm has four primary operations
  - KeyGen, Enc, Dec, and Eval
- KeyGen, Enc and Dec are essentially not different from their classical tasks in conventional encryption algorithms
  - KeyGen operation generates a secret and public key pair for an asymmetric encryption scheme and a single key for the symmetric encryption scheme
- Eval operation is the true homeomorphic encryption specific operation, it takes ciphertexts as input and outputs evaluated ciphertexts
  - Eval performs the function f() over the ciphertexts (c1, c2) without seeing the messages (m1, m2)
  - The format of the ciphertexts must be preserved after an evaluation process to be decrypted correctly
  - The size of the ciphertext should also be constant to support unlimited number of operations
    - Increase in the ciphertext size will require more resources and will limit the number of operations

RLWE-Based Homomorphic Encryption

- What LWE – Learning with Error and Ring LWE?
  - Learning with Error is a computation problem that given a set of linear equations, we solve for the secret

$$14s_1 + 5s_2 + 15s_3 + 7s_4 \approx 8 + 1 \ (mod\ 17)$$

$$3s_1 + 7s_2 + 4s_3 + 12s_4 \approx 16 + 3 \ (mod\ 17)$$

$$8s_1 + 10s_2 + 11s_3 + 3s_4 \approx 7 + 2 \ (mod\ 17)$$

$$6s_1 + 7s_2 + 16s_3 + 2s_4 \approx 3 + 4 \ (mod\ 17)$$

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- FV.SH.SecretKeyGen():
  - sample s from a Gaussian distribution, and
  - output
    - sk = s
- FV.SH.PublicKeyGen(sk):
  - set s = sk,
  - sample a from R_q and e from Gaussian distribution,
  - output
    - $pk[0] = b = [-(a.s + e)]_q$
    - pk[1] = a

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- FV.SH.Encrypt(pk, m):
  - encrypt a message $m \epsilon R_T$
  - compute $t = q/T$
  - sample r0 from R_2
  - sample r1 and r2 from the Gaussian distribution, and
  - return
    - $ct[0] = [b.r0 + r2 + t.m]_q$ and
    - $ct[1] = [a.r0 + r1]_q$

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- FV.SH.Add(ct_1, ct_2):
  - Compute $ct_1 + ct_2$
  - return
    - $c_0 = [ct_1[0] + ct_2[0]]_q$,
    - $c_1 = [ct_1[1] + ct_2[1]]_q$
- FV.SH.Decrypt(sk, ct):
  - set s = sk, and ciphertexts = c_0, c_1
  - compute $\left[\left[\frac{[c_0 + c_1.s]}{t}\right]\right]_q$
  - compute $\tilde{m}(mod\ T)$

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- FV.SH.MUL(ct_1, ct_2):
  - Compute $ct_1 * ct_2$
  - return
    - $c_0 = \left[\left\lfloor \frac{(ct_1[0] \;.\; ct_2[0])}{t} \right\rceil\right]_q$
    - $c_1 = \left[\left\lfloor \frac{(ct_1[0] \;.\; ct_2[1] + ct_1[1] \;.\; ct_2[0])}{t} \right\rceil\right]_q$
    - $c_2 = \left[\left\lfloor \frac{(ct_2[0] \;.\; ct_2[1])}{t} \right\rceil\right]_q$
- FV.SH.Decrypt(sk, ct):
  - set s = sk,
  - ciphertexts c_0, c_1 and c_2
  - compute $\tilde{m} = (c_0 s^0 + c_1 s^1 + c_2 s^2)_q$
  - compute $\tilde{m}(mod\ T)$

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- Relinearisation challenge
  - Relinearisation is a procedure that takes a degree 2 ciphertext and reduces it again to a degree 1 ciphertext
  - Let ct = $[c_0, c_1, c_2]$ denote a degree 2 ciphertext, then we need to find ct' = $[c_0', c_1']$ such that
    - $[c_0.s^0 + c_1.s^1 + c_2.s^2]_q = [c_0'.s^0 + c_1'.s^1]_q$
  - To eliminate $c_2.s^2$ term we need to mask it
    - Masking is done using relinearisation keys/ homomorphism keys/ evaluation keys

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- Noise growth challenge
  - Noise in freshly encrypted ciphertext is given by
    - ct1 has B amount of noise
    - ct2 has B amount of noise
  - SH.Add(ct1, ct2):
    - Noise growth: B + B = 2B
  - SH.Mul(ct1, ct2):
    - Noise growth: B * B = B^2
    - With L levels of multiplication
      - Noise growth: B^{2^L}

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- Noise growth challenge
  - Decryption will be correct, if
    - Noise <= q/4
  - To perform L levels of multiplication,
    - $B^{2^L} \leq q/4$ which means $q \geq 4B^{2^L}$

For B = 10,

L	q	$\log_2 q$	n
1	400	9	1024
2	40000	16	1024
3	400000000	29	2048
4	40000000000000000	56	2048

HE Computational Challenges

- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- Noise growth challenge
  - Noise in freshly encrypted ciphertext is given by
    - ct1 has B amount of noise
    - ct2 has B amount of noise
  - SH.Add(ct1, ct2):
    - Noise growth: B + B = 2B
  - SH.Mul(ct1, ct2):
    - Noise growth: B * B = B^2
    - With L levels of multiplication
      - Noise growth: B^{2^L}

HE Computational Challenges

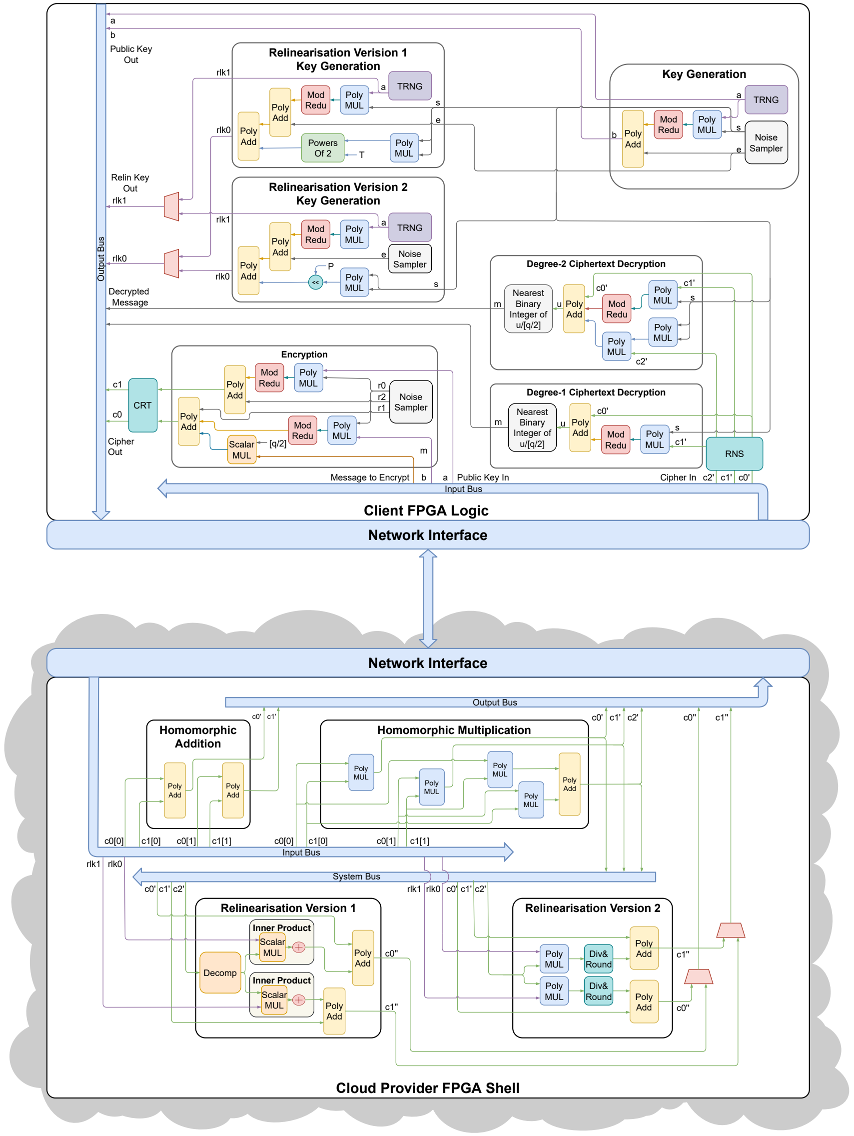
- Brakerski-Fan-Vercauteren (BFV) scheme as illustrative example
- Noise growth challenge
  - Decryption will be correct, if
    - Noise <= q/4
  - To perform L levels of multiplication,
    - $B^{2^L} \leq q/4$ which means $q \geq 4B^{2^L}$

For B = 10,

L	q	$\log_2 q$	n
1	400	9	1024
2	40000	16	1024
3	400000000	29	2048
4	40000000000000000	56	2048

Arithmetic Hardware Library for Accelerated HE

- A library consisting of modules involved in these HE algorithms
  - Residue Number System (RNS)
  - Chinese Remainder Theorem (CRT)
  - NTT-based polynomial multiplication
  - Modulo Inverse
  - Modulo Reduction
  - Polynomial and scalar operations
  - And others …

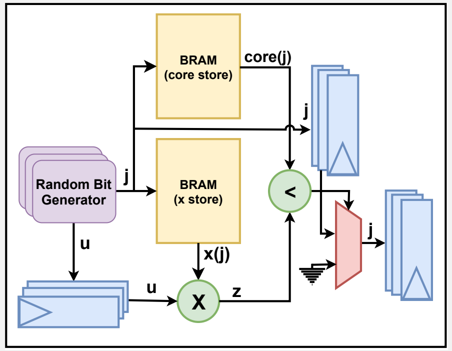


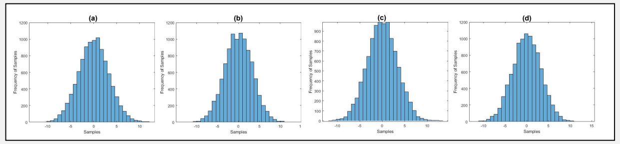
Arithmetic Hardware Library for Accelerated HE

- Gaussian Noise Sampler

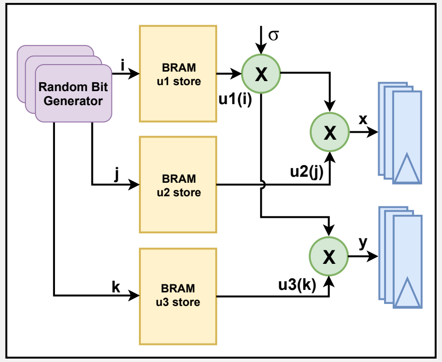
A Post-Quantum Secure Discrete Gaussian Noise Sampler Sampling Algorithms

Box-Muller Sampling





Ziggurat Sampling



HARDWARE COST FOR DIFFERENT SAMPLERS

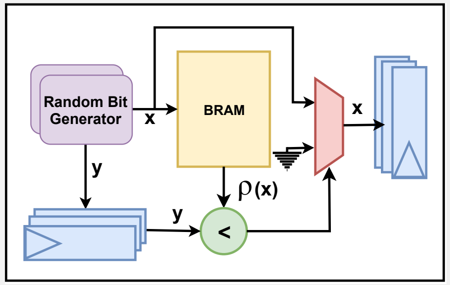
Sampling Algorithm	Slice LUTs	BRAM	DSP	Freq. (MHz)
Box-Muller Sampling	146	1	11	204.6
Rejection Sampling	89	1	0	76.4
Ziggurat Sampling	114	1.5	9	103.5

Algorithm 2 Modified Box-Muller Sampling Algorithm for Hardware Implementation

Input: σ
Output: x, y

```
Precompute:
choose u_1 ← R = ℝ ∩ [0,1] uniformly at random
choose u_2 ← R = ℝ ∩ [0,1] uniformly at random
if u_1 ≠ 0 then
    compute u1_store = sqrt(−2 ln u_1)
    compute u2_store = cos(2π u_2)
    compute u3_store = sin(2π u_2)
end if
Repeat
compute x = σ × u1_store × u2_store
compute y = σ × u1_store × u3_store
return x, y
```

Rejection Sampling



Homomorphic Encryption Libraries

- Some of the popular homomorphic encryption libraries
  - PALISADE
    - https://palisade-crypto.org/
  - SEAL
    - https://www.microsoft.com/en-us/research/project/microsoft-seal/
  - Helib
    - https://homenc.github.io/HElib/
  - HEAAN
    - https://heaan.it/
  - TFHE
    - https://tfhe.github.io/tfhe/

Upcoming Lectures

- Other Hardware Security Topics
  - Related Topics
  - Reviews