Cryptography for Cloud Security

Mohsen Toorani

Department of Informatics, University of Bergen

Simula@UiB

Coins Winter School
Finse, Norway
May 12, 2017

Our project

Title

Cryptographic Tools for Cloud Security
Funded by the Norwegian Research Council (IKTPLUSS)

Partners

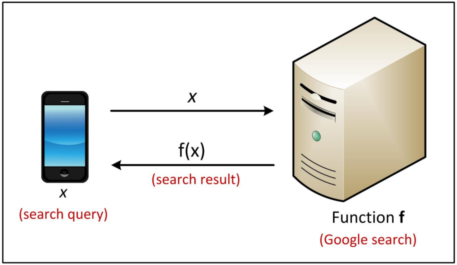
- NTNU (Department of Information Security and Communication Technology & Department of Mathematics)
- Simula@UiB

ntnu.edu/iik/cloudcrypto

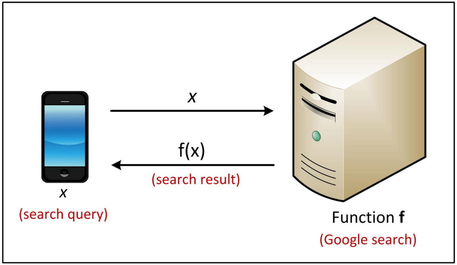
Outline

1. Computing on encrypted data
   - (Fully) Homomorphic Encryption
   - Functional Encryption
   - Obfuscation
2. Secure Deduplication
   - Deduplication schemes
   - Side channels in deduplication

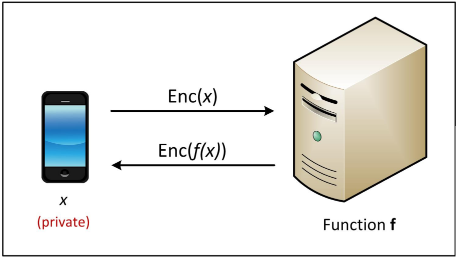
Computing on encrypted data

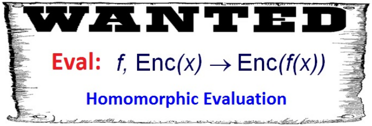


Computing on encrypted data



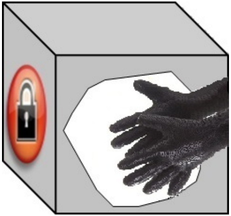
Privacy?





Homomorphic Encryption

- A way to delegate processing of data without giving access to it
- Encryption schemes that allow computations on the ciphertexts
  $E_k[m_1] \bullet E_k[m_2] = E_k[m_1 \circ m_2]$
- Applications:
  - E-voting: Votes are encrypted as 1 or 0. Ciphertexts are aggregated before decryption. No individual vote is revealed.
    Requires additive homomorphic encryption: $\circ$ is $+$
  - Secure cloud computing: Requires fully homomorphic encryption (homomorphic properties for both $+$ and $\times$)



Homomorphic Encryption

Multiplicative homomorphic encryption

- Unpadded RSA: $m_1^e \times m_2^e = (m_1 \times m_2)^e$
- ElGamal: Given public key $(g, h = g^a)$, ciphertexts $(g^{r_1}, h^{r_1} m_1)$ and $(g^{r_2}, h^{r_2} m_2)$, multiple both components $(g^{r_1+r_2}, h^{r_1+r_2} m_1 m_2)$

Additive homomorphic encryption

Paillier cryptosystem [Eurocrypt'99]: Additive on $\mathbb{Z}_n$

- Public key: (n, g) where p and q: two large prime, $n = pq$, $g \in_R \mathbb{Z}^*_{n^2}$
- Private key: (λ, μ) where $\lambda = lcm(p-1, q-1)$, and $\mu = (\frac{g^\lambda mod n^2 - 1}{n})^{-1} mod n$
- For encrypting $m \in \mathbb{Z}_n$: Select random $r \in_R \mathbb{Z}^*_n$
  Compute $c = g^m r^n \mod n^2$
- For decryption: compute $m = \mu \frac{c^\lambda mod n^2 - 1}{n} \mod n$

Homomorphic Encryption
Continued

Examples of schemes with limited functionality

- RSA works for MULT (mod N)
- Paillier works for ADD (XOR)
- BGN05 works for quadratic formulas
- MGH08 works for low-degree polynomials
  (size of $c \leftarrow Eval(pk, f, c_1, ..., c_t)$ grows exponentially with degree of f)

Somewhat Homomorphic Encryption (SHE)

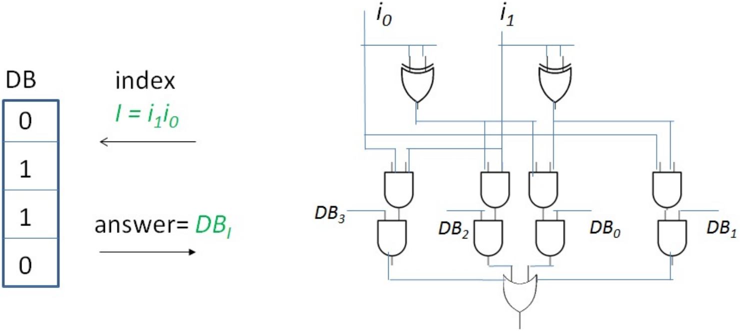
- Eval only works for some functions f

Fully Homomorphic Encryption (FHE)

- Fully means that it works for any arbitrary function f
- Supports both addition and multiplication
- Before Gentry's work (2009), no FHE scheme

Why both addition and multiplication?

- Because {XOR, AND} is Turing-complete: any function can be written as a combination of XOR and AND gates.
- If you can compute XOR and AND on encrypted bits, you can compute ANY function on encrypted inputs.



Example: Searching a database

Homomorphic Public-key Encryption

- Procedures: (KeyGen, Enc, Dec, Eval)

  $(sk, pk) \leftarrow KeyGen(\lambda)$

- Correctness: For any function f in supported family F,

  $c_1 \leftarrow Enc_{pk}(m_1)$, ... , $c_t \leftarrow Enc_{pk}(m_t)$

  $c^* \leftarrow Eval_{pk}(f, c_1, ..., c_t)$

  $Dec_{sk}(c^*) = f(m_1, ..., m_t)$

- No information about m_1, ..., m_t, and $f(m_1, ..., m_t)$ is leaked.

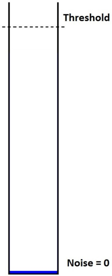
- Compactness: complexity of decrypting c^* does not depend on complexity of f.

SHE + Bootstrappability → FHE

1. Construct a useful "Somewhat Homomorphic Encryption" scheme
2. Modify your SHE scheme and make it bootstrappable if it is not
3. Bootstrappable SHE $\xrightarrow[\text{Recryption}]{\hspace{3cm}}$ FHE

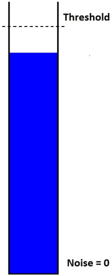
(Note: It is also possible to construct FHE schemes without bootstrapping).

Bootstrapping



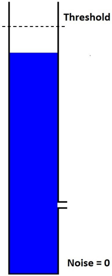
- Problem: Ciphertexts contain random 'noise' that grows after homomorphic evaluation (Add and Mult increase noise).
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Without a noise-reduction, number of homomorphic operations that can be performed is limited.
- The best noise-reduction that kills all noise: Decryption!
- Decryption should be done without releasing the secret key
  $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
  (For a cycle of public/secret key-pairs (pk_i, sk_i) for $i = 1, ..., n$, encrypt each sk_i under $pk_{(i \mod n)+1}$.)
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping $=$ "Valve" at a fixed height
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].
- Bootstrapping requires homomorphically evaluating the decryption circuit.

Bootstrapping



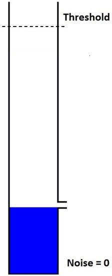
- Problem: Ciphertexts contain random 'noise' that grows after homomorphic evaluation (Add and Mult increase noise).
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Without a noise-reduction, number of homomorphic operations that can be performed is limited.
- The best noise-reduction that kills all noise: Decryption!
- Decryption should be done without releasing the secret key
  $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
  (For a cycle of public/secret key-pairs (pk_i, sk_i) for $i = 1, \ldots, n$, encrypt each sk_i under $pk_{(i \mod n)+1}$.)
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping $=$ "Valve" at a fixed height
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].
- Bootstrapping requires homomorphically evaluating the decryption circuit.

Bootstrapping



- Problem: Ciphertexts contain random 'noise' that grows after homomorphic evaluation (Add and Mult increase noise).
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Without a noise-reduction, number of homomorphic operations that can be performed is limited.
- The best noise-reduction that kills all noise: Decryption!
- Decryption should be done without releasing the secret key
  $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
  (For a cycle of public/secret key-pairs (pk_i, sk_i) for $i = 1, ..., n$, encrypt each sk_i under $pk_{(i \mod n)+1}$.)
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping $=$ "Valve" at a fixed height
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].
- Bootstrapping requires homomorphically evaluating the decryption circuit.

Bootstrapping



- Problem: Ciphertexts contain random 'noise' that grows after homomorphic evaluation (Add and Mult increase noise).
- Once the noise exceeds a certain level, the ciphertext can no longer be decrypted.
- Without a noise-reduction, number of homomorphic operations that can be performed is limited.
- The best noise-reduction that kills all noise: Decryption!
- Decryption should be done without releasing the secret key
  $\rightarrow$ We can release $Enc(sk)$: Circular Encryption
  (For a cycle of public/secret key-pairs (pk_i, sk_i) for $i = 1, ..., n$, encrypt each sk_i under $pk_{(i \mod n)+1}$.)
- Whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level. Bootstrapping = "Valve" at a fixed height
- Gentry's "bootstrapping" theorem: If an encryption scheme can evaluate its own decryption circuit, then it can evaluate everything [Gentry'09].
- Bootstrapping requires homomorphically evaluating the decryption circuit.

Recryption

- A central aspect in Gentry's FHE (and subsequent schemes).
- It allows to refresh a ciphertext: given a ciphertext C, compute a new ciphertext C' with a decreased noise.
- By periodically refreshing the ciphertext (e.g., after computing some gates in f), one can evaluate arbitrarily large circuits f.
- Recryption is implemented by evaluating the decryption circuit of the encryption scheme homomorphically.

Homomorphic Public-key Encryption

Semantic security

- Procedures: (KeyGen, Enc, Dec, Eval)
- Semantic security is defined like basic encryption.
- Notions of security in basic public-key encryption schemes:

$$
\begin{tikzcd}
\text{NM-CPA} \arrow[d] & \text{NM-CCA1} \arrow[l] \arrow[d] & \text{NM-CCA2} \arrow[l] \arrow[d, shift right] \\
\text{IND-CPA} & \text{IND-CCA1} \arrow[l] & \text{IND-CCA2} \arrow[l] \arrow[u, shift right]
\end{tikzcd}
$$

- Malleability of ciphertexts $\rightarrow$ Homomorphic encryption cannot achieve IND-CCA2.
  Non-malleability (NM): an adversary's inability to transform a given ciphertext into a different ciphertext so that their according plaintexts are "meaningfully related".
- FHE schemes that adopt Gentry's bootsrapping technique might not be CCA1-secure.

Hard Problems

For constructing homomorphic encryption schemes

- Shortest Vector Problem (SVP): shortest possible vector in the lattice
- Closest Vector Problem (CVP): closest vector to a point
- Learning With Errors (LWE): a generalization to "*parity with noise*" problem
  - Polynomial Learning With Errors (PLWE)
  - Ring Learning With Errors (RLWE)
- Sparse Subset Sum Problem (SSSP)
- Bounded Distance Decoding (BDD)
- Approximate Greatest Common Divisor (AGCD)
- Polynomial Coset Problem (PCP): related to Ideal Coset Problem

Multi-key FHE

- Different clients encrypt data under different FHE keys.
- The cloud combines data encrypted under different keys:
  $Enc_{pk_1,...,pk_t}(f(m_1, ..., m_t)) \leftarrow Eval(pk_1, ..., pk_t, f, c_1, ..., c_t)$
- FHE does not provide it automatically.
- It is possible to construct FHE schemes with above property:
  [LATV12] "On-the-fly Multiparty Computation on the Cloud via Multi-key FHE."

A Construction of FHE [DGHV'10]

1. Construct a Symmetric Somewhat Homomorphic Encryption
(under the approximate GCD assumption)
2. By a simple transformation, convert it to a Public-key Somewhat Homomorphic Encryption
(under the approximate GCD assumption)
3. Use Gentry's techniques to have a public-key FHE
(under approximate GCD + sparse subset sum)

Approximate GCD Problem

- Given many $x_i = s_i + q_i p$, output p
- Example parameters: $s_i \sim 2^{\lambda}$, $p \sim 2^{\lambda^2}$, $q_i \sim 2^{\lambda^5}$
(λ: security parameter)
- Best known attacks (lattice-based): $\sim 2^{\lambda}$ time

A Construction of FHE [DGHV'10]

Step 1: Constructing a symmetric homomorphic encryption scheme

Secret key

large odd number p

Encryption steps of a bit m

Choose at random large q and small r

$c = pq + 2r + m$

If $2r + m \ll p$ then ciphertext is close to a multiple of p

Parameters: $|r| = n$, $|p| = n^2$ and $|q| = n^5$

Decryption

$m \equiv (c \mod p) \mod 2$

Why is it homomorphic?

$c_1 = pq_1 + 2r_1 + m_1, \qquad c_2 = pq_2 + 2r_2 + m_2$

- $c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + (m_1 + m_2)$
  If $(r_1 + r_2) \ll \frac{p}{2} \Rightarrow (c_1 + c_2 \mod p) \mod 2 \equiv m_1 + m_2 (mod 2)$
  Noise $= 2 \times$ (Initial noise)
- $c_1 c_2 = (q_1 q_2 p + 2q_1 r_2 + q_1 m_2 + 2q_2 r_1 + q_2 m_1)p + 2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 m_2$
  If $(2r_1 r_2 + r_1 m_2 + m_1 r_2) \ll \frac{p}{2} \Rightarrow (c_1 c_2 \mod p) \mod 2 \equiv m_1 m_2 (mod 2)$
  Noise $=$ (Initial noise)^2

Comparison of Fully Homomorphic Encryption Schemes

Scheme	Year	Underlying Problems	Asymptotic Runtime	Concrete Runtime
Gentry: A Fully Homomorphic Encryption Scheme	2009	BDDP + SSSP	$O(\lambda^{3.5})$ per gate for ciphertext refreshing	NA
van Dijk, Gentry, Halevi, Vaikuntanathan: FHE over the Integers	2010	AGCD + SSSP	Public key size: $O(\lambda^{10})$, no gate cost given	NA
Smart, Vercauteren: FHE with Relatively Small Key and Ciphertext Sizes	2010	PCP + SSSP	Key generation: $O(\log n.n^{2.5})$	Key generation: several hours even for small parameters, for larger parameters the keys could not be generated
Brakerski, Vaikuntanathan: Efficient FHE from (standard) LWE	2011	DLWE	Evaluation key size: $O(\lambda^{2C}\log(\lambda))$	-
Brakerski, Vaikuntanathan: FHE from Ring-LWE and Security for Key Dependent Messages	2011	PLWE	Very cheap key generation, unknown for bootstrapping	-
Brakerski, Gentry, Vaikuntanathan: FHE without Bootstrapping	2011	RLWE	Per-gate computation overhead $O(d^3\lambda\log\lambda)$ without bootstrapping, $O(\lambda^2\log\lambda)$ with bootstrapping	36 hours for an AES encryption on a supercomputer

d: Depth of the circuit, n: Dimension of the lattice, C: A very large parameter for ensuring bootstrappability

Comparison of Fully Homomorphic Encryption Schemes

Continued

Gentry, Halevi: Implementing Gentry's Fully-Homomorphic Encryption Scheme	2011	SVP + BDD	Key generation: $O(\log n.n^{1.5})$	Bootstrapping: From 30s (for small setting) to 30 min (for large setting)
Coron, Naccache, Tibouchi: Public Key Compression and Modulus Switching for FHE over the Integers	2012	DAGCD + SSSP	Public key size: $O(\lambda^5 \log(\lambda))$, no gate cost given	Recryption: 11 min
Rohloff, Cousins: A Scalable Implementation of Fully Homomorphic Encryption Built on NTRU	2014	SVP + RLWE	-	Recryption: 275s on 20 cores with 64-bit security
Halevi, Shoup: Bootstrapping for HElib	2015	RLWE	-	Vectors of 1024 elements from $GF(2^{16})$ was recrypted in 5.5 min at security level ≈ 76, single CPU core

Table From: Armknecht et al. [ABCGJRS'15]

Fewer Multiplications?

Symmetric ciphers for FHE, MPC, etc

- FHE schemes typically come with a ciphertext expansion in the order of 1000 to 1000000.
- Compression method: instead of sending $c = HE_{pk}(m)$ to the cloud, pick a random key k and send $c' = (HE_{pk}(k), E_k(m))$.
- For long messages, $|c'|/|m| \approx 1$.
- By homomorphically evaluating the decryption circuit $\mathcal{C}_{E^{-1}}$, the cloud recovers $c = HE_{pk}(m) = \mathcal{C}_{E^{-1}}(HE_{pk}(k); E_k(m))$.
- Symmetric encryption algorithms for FHE, MPC, etc:
  - LowMC (block cipher): eprint 2016/687
  - Kreyvium (stream cipher): eprint 2015/113
  - FLIP (stream cipher): eprint 2016/254

Functional Encryption

A public key FE scheme for a class of circuits $\mathcal{C}_\lambda$ is a tuple of PPT algorithms (FE.Setup, FE.KeyGen, FE.Enc, FE.Dec) [O'N10, BSW12]):

- $(msk, pk) \leftarrow FE.Setup(1^\lambda)$: FE.Setup takes as input the security parameter λ and outputs the master secret key msk and public key pk.
- $sk_C \leftarrow FE.KeyGen(msk, C)$: FE.KeyGen takes as input the master secret key and a circuit $C \in \mathcal{C}_\lambda$ and outputs the functional secret key sk_C.
- $c \leftarrow FE.Enc(pk, m)$: FE.Enc takes as input the public key and message $m \in \{0, 1\}^*$ and outputs the ciphertext c.
- $y \leftarrow FE.Dec(sk_C, c)$: FE.Dec takes as input the functional secret key and ciphertext and outputs $y \in \{0, 1\}^*$.

FHE and FE

- FHE: compute $Enc(f(x))$ from $Enc(x)$ for any function f.
- FE: compute $f(x)$ from $Enc(x)$.
- For functions of the type Enc_f, where $Enc_f(x) = Enc(f(x))$ is a re-encryption of $f(x)$, FE would be very close to constructing an FHE scheme.
- Randomized FE can be used for constructing FHE [ABFGGTW'13].
- Randomized FE (FE with randomized functionality): privacy-aware auditing, differentially private data release, proxy re-encryption, ...
- rFE is not much more difficult to construct than the standard FE.

Obfuscation

- Program obfuscation: to scramble a computer program, hiding its implementation details (making it hard to reverse-engineer), while preserving the functionality (i.e, input-output behaviour) of the program.
- Obfuscation:
  - The cloud is given an "encrypted" program $E(P)$.
  - For any input x, cloud can compute $E(P)(x) = P(x)$.
  - Cloud learns nothing about P, except $\{x_i, P(x_i)\}$.

iO

- Notion of indistinguishability obfuscation (iO) has emerged as the central notion of obfuscation.
- iO requires that obfuscations $iO(C_1)$, $iO(C_2)$ of any two functionally equivalent circuits C_1 and C_2 (i.e., whose outputs agree on all inputs) from some class of bounded-size circuits $\mathcal{C}$ are computationally indistinguishable.
- All candidate constructions of iO rely on candidate constructions of multilinear maps, all of which have non-trivial attacks.

Formal definition

A uniform PPT machine is called an iO for a circuit class $\{\mathcal{C}_\lambda\}$ if

- Correctness: For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$ and all inputs x, we have $Pr[C'(x) = C(x) \mid C' \leftarrow iO(\lambda, C)] = 1$
- Security: For any PPT distinguisher D, there exists a negligible function ϵ such that for all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have if $C_0(x) = C_1(x)$ for all inputs x then $| Pr[D(iO(\lambda, C_0)) = 1] - Pr[D(iO(\lambda, C_1)) = 1] | \leq \epsilon(\lambda)$

FHE and Obfuscation

- FHE does not provide obfuscation automatically.
- It is possible to use obfuscated circuits to obtain Randomized Functional Encryption schemes suitable for FHE constructions [ABFGGTW'13]:
  Obfuscated circuits $\rightarrow$ Randomized Functional Encryption $\rightarrow$ FHE

Inefficient iO

- It is important that the obfuscator is efficient (polynomial-time).
- A trivial inefficient iO with running time $poly(|C|, \lambda).2^n$ exists unconditionally: Simply output the function table of C (i.e. the output of C on all possible inputs).
  C: the circuit to be obfuscated
  λ: security parameter
  n: input length of C
- In "standard" (efficient) iO, the running time and size of the obfuscator is required to be $poly(|C|, \lambda)$ (polylogarithmic in the size of the truth table of C).

XiO

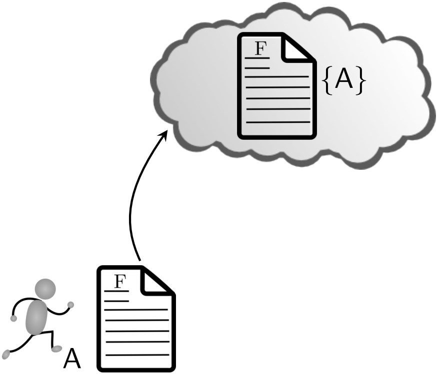
- XiO: Exponentially-Efficient iO
  Lin et al., Indistinguishability Obfuscation with Non-trivial efficiency, (PKC'16), eprint 2016/006
- Inefficient iO with running time $poly(|C|, \lambda).2^n$ exists unconditionally.
- In XiO, running-time of the obfuscator may still be trivial ($poly(|C|, \lambda).2^n$), but the obfuscated code is just slightly smaller than the truth table of C ($poly(|C|, \lambda).2^{n(1-\epsilon)}$ where $\epsilon > 0$).
- *Succinct FE*: A compact FE for a class of circuits that output only a single bit.
- There is NOT any black-box deduction from succinct FE to iO
- There is black-box deduction from succinct FE to XiO

Other topics

- Multiparty Computation (MPC)
- Delegation of Computation
- Searchable Encryption
- Attribute-based Encryption
- ...

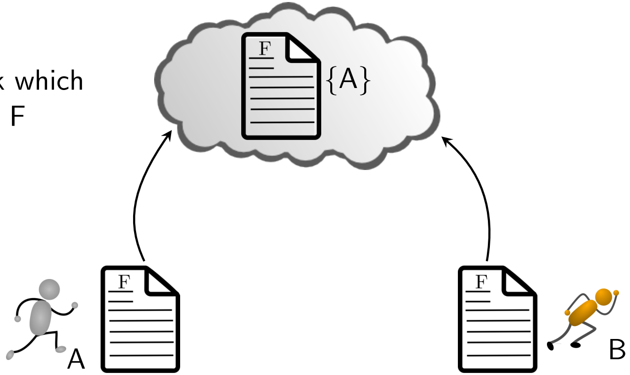
Secure Deduplication

Cloud Storage



Cloud Storage

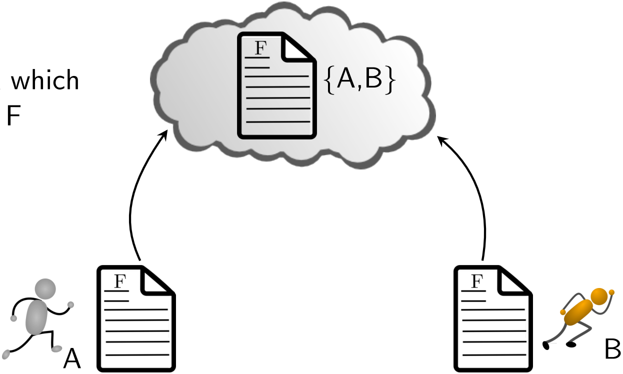
Server needs to track which users have access to F



Cloud Storage

Server needs to track which users have access to F

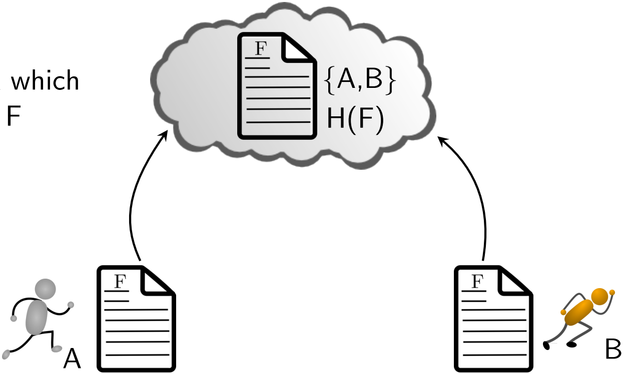
When B uploads F, server updates tag



Cloud Storage

Server needs to track which users have access to F

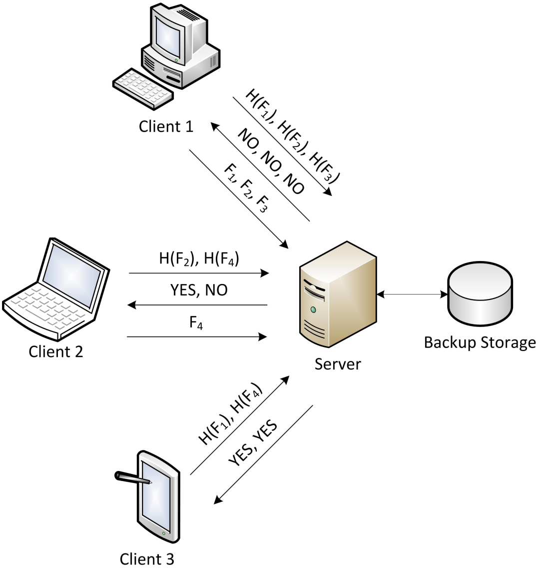
When B uploads F, server updates tag



Smarter: client-side deduplication where users send H(F)

[Inside Dropbox, Drago et al., IMC'12]

Simple client-side deduplication



Deduplication

- **Deduplication**: store only a single copy of each file (or block)
- Can save more than 90% of storage in many business scenarios (major savings in media files and software)
- Server-side deduplication: Data is always uploaded, but only one copy is stored on the cloud.
  - + saves storage
- Client-side deduplication: Data is uploaded if it is not available on the cloud.
  - + saves storage
  - + saves bandwidth

- Encryption is at odds with cross-user deduplication: solutions exist to derive key from file itself [MLE, MLE2, iMLE, Dupless, PAKE-based, ...]
- Serious privacy concerns may arise when deduplication is used by popular storage services (side-channels).

Convergent Encryption

Douceur et al. (ICDCS'02)

- P_f: File plaintext

  E: Symmetric key encryption

  F: Public key encryption

  (K_u, K'_u): Public/private key pair for each user u

- $C_f = X_{K_u}(P_f) =< c_f, M_f >$ in which
  - $c_f = E_{H(P_f)}(P_f)$
  - $M_f = \{\mu_u = F_{K_u}(H(P_f)) \wedge u \in U_f\}$

- $P_f = X^{-1}_{K'_u}(C_f) = E^{-1}_{F^{-1}_{K'_u}(\mu_u)}(c_f)$

MLE

Bellare et al. (EUROCRYPT'13)

- Alice:
  - $k_A \leftarrow K(P, m_A)$
  - $c_A \leftarrow E(k_A, m_A)$
  - $t_A \leftarrow T(P, c_A)$
  - upload c_A to the server
- Bob:
  - $k_B \leftarrow K(P, m_B)$
  - $c_B \leftarrow E(k_B, m_B)$
  - $t_B \leftarrow T(P, c_B)$
  - upload c_B to the server
- Tag correctness: if $t_A = t_B$ then $m_A = m_B$ and the server deduplicates.
- Duplicate faking attack: if $t_A = t_B$ but $m_A \neq m_B$ (integrity violation).
- CE is a special case of the MLE where $k = H(m)$ and $T = H(c)$.

Can we get IND-CPA style privacy for MLE?

Message recovery security

Consider a set $S = \{m_1, ..., m_n\}$
Given $c \leftarrow E(K(m_i), m_i)$ where $i \leftarrow \{1, 2, \ldots, n\}$
Find m_i

$BruteForce_S(c)$

For $m_i \in S$:
$m' \leftarrow D(K(m_i), c)$
If $m_i = m'$ then return m_i

Privacy is not possible for predictable messages.
MLE schemes cannot achieve semantic-security-style privacy.

MLE

Notions for Privacy

$$
\begin{tikzcd}
\text{PRV\$-CDA} \arrow[rr] \arrow[d, shift right] & & \text{PRV\$-CDA-A} \arrow[d, shift right] \\
\text{PRV-CDA} \arrow[rr, "/" marking] \arrow[u, shift right, "/" marking] & & \text{PRV-CDA-A} \arrow[u, shift right, "/" marking]
\end{tikzcd}
$$

CDA: Chosen-Distribution Attack

PRV-CDA: Encryptions of two unpredictable messages should be indistinguishable (for non-adaptive adversaries).

PRV\$-CDA: Encryption of an unpredictable message must be indistinguishable from a random string of the same length (for non-adaptive adversaries).

Tag Consistency (TC): hard to create (M, C) s.t.:
$T(C) = T(E(K(M), M))$ but $D(K(M), C) = M' \neq M$

Strong Tag Consistency (STC): hard to create (M, C) s.t.:
$T(C) = T(E(K(M), M))$ but $D(K(M), C) = \perp$

TC & STC: Preserve integrity

MLE schemes in RO

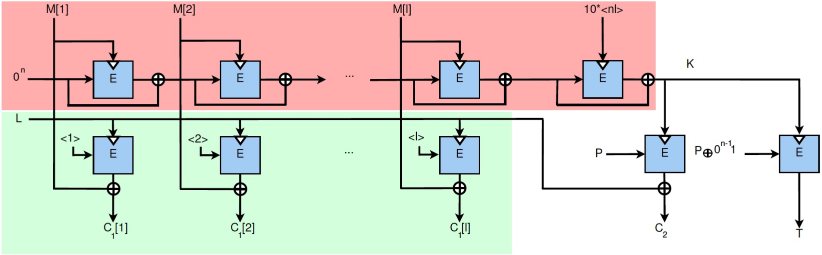
Scheme	Model	D/R	Privacy		Integrity	
			PRV-CDA	PRV$-CDA	TC	STC
CE	RO	D	✓	✓	✓	✓
HCE1	RO	D	✓	✓	✗	✗
HCE2	RO	D	✓	✓	✓	✗
RCE	RO	R	✓	✓	✓	✗

- CE: $K = H(M)$, $C = E(K, M)$, $T = H(C)$
- HCE1: $K = H(M)$, $C = E(K, M)||H(K)$, $T = H(K)$
- HCE1 does not provide TC (vulnerable to duplicate faking attack).

MLE schemes
in RO

Scheme	Key generation $\mathcal{K}_P(M)$	Encrypt $\mathcal{E}_P(K, M)$	Tag generation $\mathcal{T}_P(C)$	Decrypt $\mathcal{D}_P(K, C)$
CE[SE, H] Convergent Encryption	$K \leftarrow \mathcal{H}(P, M)$ Ret K	$C \leftarrow \mathcal{SE}(K, M)$ Ret C	Ret $\mathcal{H}(P, C)$	Ret $\mathcal{SD}(K, C)$
HCE1[SE, H] Hash and CE w/o tag check		$T \leftarrow \mathcal{H}(P, K)$ $C \leftarrow \mathcal{SE}(K, M)$ Ret $C \parallel T$	Parse C as $C_1 \parallel T$ Ret T	Parse C as $C_1 \parallel T$ $M \leftarrow \mathcal{SD}(K, C)$ Ret M
HCE2[SE, H] Hash and CE w/ tag check				Parse C as $C_1 \parallel T$ $M \leftarrow \mathcal{SD}(K, C)$ $T' \leftarrow \mathcal{H}(P, \mathcal{H}(P, M))$ If $T' \neq T$ then $\perp$ Ret M
RCE[SE, H] Randomized Convergent Encryption		$L \leftarrow_\$ \{0,1\}^{k(\lambda)}$ $T \leftarrow \mathcal{H}(P, K)$ $C_1 \leftarrow \mathcal{SE}(L, M)$ $C_2 \leftarrow L \oplus K$ Ret $C_1 \parallel C_2 \parallel T$	Parse C as $C_1 \parallel C_2 \parallel T$ Ret T	Parse C as C_1, C_2, T $L \leftarrow C_2 \oplus K$ $M \leftarrow \mathcal{SD}(L, C_1)$ $T' \leftarrow \mathcal{H}(P, \mathcal{H}(P, M))$ If $T' \neq T$ then Ret $\perp$ Ret M

MLE schemes built using symmetric encryption scheme $SE = (SK; SE; SD)$ and hash function family $H = (HK; H)$.



$\underline{\mathcal{P}(1^\lambda)}$
$P \leftarrow\!\!\$\, \{0,1\}^n$; Ret P

$\underline{\mathcal{K}_P(M)}$
$K \leftarrow \mathrm{MD}[E](M)$; Ret K

$\underline{\mathcal{E}_P(K,M)}$
$L \leftarrow\!\!\$\, \{0,1\}^n$
$C_1 \leftarrow \mathsf{CTR}[E](L,M)$
$C_2 \leftarrow L \oplus E(K,P)$
$T \leftarrow E(K, P \oplus 0^{n-1}1)$
Ret (C_1, C_2, T)

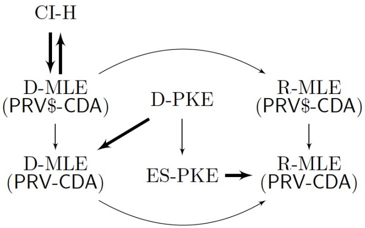
$\underline{\mathcal{D}_P(K,(C_1,C_2,T))}$
$L \leftarrow C_2 \oplus E(K,P)$
$M \leftarrow \mathsf{CTR}[E](L,C_1)$
$K' \leftarrow \mathrm{MD}[E](M)$
If $E(K', P \oplus 0^{n-1}1) = T$ then Ret M
Else Ret $\perp$

- CE[E]: Set $L = 0^n$, use C_2 as the key for CTR[E], exclude C_2 from the ciphertext, tag generation hashes the ciphertext: E(MD[E](M); P).
- HCE2[E]: Use C_2 as the encryption key for CTR[E]. Exclude C_2 from the ciphertext.

MLE

Constructions without RO

Scheme	Model	D/R	Privacy		Integrity	
			PRV-CDA	PRV$-CDA	TC	STC
CE	RO	D	✓	✓	✓	✓
HCE1	RO	D	✓	✓	✗	✗
HCE2	RO	D	✓	✓	✓	✗
RCE	RO	R	✓	✓	✓	✗
XtCIH	STD	D	✓	✓	✓	✓
XtDPKE	STD	D	✓	✗	✓	✓
XtESPKE	STD	R	✓	✗	✓	✓
SXE	STD	D	✓	✓	✓	✓



D-PKE: Deterministic Public-Key Encryption

CI-H: Correlated-input-secure Hash Function

MLE for Lock-dependent messages: MLE2

Abadi et al. (CRYPTO'13)

- Strengthened the notions of security by considering plaintext distributions that may depend on the public parameters of the schemes (lock-dependent messages).
- First (main) construction (A fully randomized scheme): R-MLE2
  - tag $\tau = (g^r, g^{rh(m)})$
  - Suppose $\tau_1 = (g_1, h_1) = (g^{r_1}, g^{r_1 h(m_1)})$ and $\tau_2 = (g_2, h_2) = (g^{r_2}, g^{r_2 h(m_2)})$
  - Equality testing: $e(g_1, h_2) \stackrel{?}{=} e(g_2, h_1)$
  - If $e(g^{r_1}, g^{r_2 h(m_2)}) = e(g^{r_2}, g^{r_1 h(m_1)})$ then $h(m_1) = h(m_2)$.
- Second construction has a deterministic ciphertext component for more efficient equality testing.

iMLE

Bellare and Keelveedhi (PKC'15)

- iMLE: Interactive message-locked encryption
- Using interaction, it provides security for messages that are both correlated and dependent on the public system parameters.
- They first construct a seemingly weak primitive: MLE-Without-Comparison (MLEWC)
- To enable comparison between ciphertexts, they introduce FCHECK which employs an interactive protocol based on a FHE scheme which transforms the MLEWC into iMLE.
- FCHECK is a theoretical construction.

Scheme(s)	Type	Messages		STD/ROM
		Correlated	Param. dep.	
CE, HCE1, HCE2, RCE [12]	MLE	Yes	No	ROM
XtDPKE, XtESPKE, ... [12]	MLE	Yes	No	STD
BHK [10]	MLE	Yes	No	STD
ABMRS [1]	MLE	No	Yes	RO
FCHECK	iMLE	**Yes**	**Yes**	**STD**

Other schemes
for secure deduplication

- DupLESS (USENIX Security 2013): Introduces a third-party key server and uses MLE. Each client engages with the key server in an oblivious pseudo-random protocol (OSRP) to obtain a message-derived key.
- Stanek et al. (FC'14): Notion of popularity + threshold cryptosystem
- PerfectDedup (DPM'15): Considers notion of popularity for block-level deduplication. Uses convergent encryption and perfect hashing.
- μR-MLE2 (ACISP'16): reduces the overhead of R-MLE2 by using static and dynamic decision trees.
- ...

Other related topics

- Proof-of-ownership (PoW): enables a client to prove the possession of a file (rather than just some short information about it).
- Proof-of-Retrievability (PoR): Interactive protocols that cryptographically prove the retrievability of outsourced data.

Side channels in deduplication

joint work with Frederik Armknecht, Colin Boyd, Gareth T. Davies, and Kristian Gjøsteen

eprint 2016/977

Inherent Side Channel

Classical client-side deduplication: server only asks client to send file if not already stored by the server

Inherent Side Channel

Classical client-side deduplication: server only asks client to send file if not already stored by the server

$\Rightarrow$ side channel ("existence-of-file attack")

Adversarial client can learn if (low-entropy) files are stored or not

Inherent Side Channel

Classical client-side deduplication: server only asks client to send file if not already stored by the server

$\Rightarrow$ side channel ("existence-of-file attack")

Adversarial client can learn if (low-entropy) files are stored or not

Identifying files, learning file contents, covert channels, ...

Examples: Clinical lab test results, figures in tax returns, pay stubs and contracts, bank letters including password or PIN, ...

Randomized solution

Idea: To use a randomized threshold for each file

[Harnik et al., Side Channels in Cloud Services: Deduplication in Cloud Storage, IEEE Security and Privacy Magazine, 2010]

Randomized solution

Idea: To use a randomized threshold for each file
[Harnik et al., Side Channels in Cloud Services: Deduplication in Cloud Storage, IEEE Security and Privacy Magazine, 2010]

Denote as thr the number of uploads before the server informs clients that it has enough copies.

Randomized solution

Idea: To use a randomized threshold for each file
[Harnik et al., Side Channels in Cloud Services: Deduplication in Cloud Storage, IEEE Security and Privacy Magazine, 2010]

Denote as thr the number of uploads before the server informs clients that it has enough copies.
If thr is chosen uniformly from the range $\{1, \ldots, \mathrm{B}\}$ for some integer B then an adversary launching the existence-of-file attack will learn nothing if thr $\in \{2, \ldots, \mathrm{B} - 1\}$.

Randomized solution

Idea: To use a randomized threshold for each file
[Harnik et al., Side Channels in Cloud Services: Deduplication in Cloud Storage, IEEE Security and Privacy Magazine, 2010]

Denote as thr the number of uploads before the server informs clients that it has enough copies.
If thr is chosen uniformly from the range $\{1, \ldots, B\}$ for some integer B then an adversary launching the existence-of-file attack will learn nothing if thr $\in \{2, \ldots, B-1\}$.
The expected number of uploads of a file is $\frac{B+1}{2}$.

Deduplication Strategies

A deduplication strategy DS is characterized by its probability distribution $\mathsf{DS}(F, \lambda) = (p_1(F, \lambda), p_2(F, \lambda), \ldots)$ where

$$p_i(F, \lambda) = \mathbf{Pr}\left[i \leftarrow \mathsf{DS.Alg}(F, \lambda)\right]$$

A threshold selection algorithm DS.Alg is a probabilistic procedure that outputs a threshold thr $\in \mathbb{N}$:

$$\mathsf{thr} \leftarrow \mathsf{DS.Alg}(\mathsf{F}, \lambda)$$

Deduplication Strategies

A deduplication strategy DS is characterized by its probability distribution $\mathsf{DS}(F,\lambda) = (p_1(F,\lambda), p_2(F,\lambda), \dots)$ where

$$p_i(F,\lambda) = \mathbf{Pr}\left[i \leftarrow \mathsf{DS.Alg}(F,\lambda)\right]$$

A threshold selection algorithm DS.Alg is a probabilistic procedure that outputs a threshold $\mathsf{thr} \in \mathbb{N}$:

$$\mathsf{thr} \leftarrow \mathsf{DS.Alg}(\mathsf{F},\lambda)$$

DS is file-oblivious if the distributions are independent of the file:

$$\mathsf{DS.Alg}(F,\lambda) = \mathsf{DS.Alg}(F^*,\lambda), \quad \forall\lambda \in \mathbb{N},\ \forall F, F^* \in \{0,1\}^*$$

Deduplication Strategies

A deduplication strategy DS is characterized by its probability distribution $\mathrm{DS}(F,\lambda) = (p_1(F,\lambda), p_2(F,\lambda), \dots)$ where

$$p_i(F,\lambda) = \mathbf{Pr}\left[i \leftarrow \mathrm{DS.Alg}(F,\lambda)\right]$$

A threshold selection algorithm DS.Alg is a probabilistic procedure that outputs a threshold thr $\in \mathbb{N}$:

$$\mathrm{thr} \leftarrow \mathrm{DS.Alg(F},\lambda)$$

DS is file-oblivious if the distributions are independent of the file:

$$\mathrm{DS.Alg}(F,\lambda) = \mathrm{DS.Alg}(F^*,\lambda), \quad \forall\lambda \in \mathbb{N},\ \forall F, F^* \in \{0,1\}^*$$

DS is finite if for $\forall\lambda$ and $\forall F$, there exists an upper bound $\mathrm{B} = \mathrm{B}(F,\lambda)$ such that $p_j(F,\lambda) = 0$ for $\forall j > \mathrm{B}(F,\lambda)$.

Deduplication Strategies

For each file, server employs deduplication strategy $\mathsf{DS} = (p_1, p_2, p_3, \dots)$

$p_0 = 0$

$$\textsf{thr} \leftarrow \mathsf{DS.Alg}(\mathsf{F}, \lambda)$$

Deduplication Strategies

For each file, server employs deduplication strategy $\mathsf{DS} = (p_1, p_2, p_3, \dots)$

$p_0 = 0$

$$\mathsf{thr} \leftarrow \mathsf{DS.Alg}(\mathsf{F}, \lambda)$$

$\Rightarrow$ server asks for file thr times before saying it already has it

Deduplication Strategies

For each file, server employs deduplication strategy $\mathsf{DS} = (p_1, p_2, p_3, \dots)$

$p_0 = 0$

$$\mathsf{thr} \leftarrow \mathsf{DS.Alg}(\mathsf{F}, \lambda)$$

$\Rightarrow$ server asks for file thr times before saying it already has it

Examples:

- A server that does not defend against the existence-of-file attack: $\mathsf{DS}^{\mathsf{dnd}} = (1, 0, \dots)$

Deduplication Strategies

For each file, server employs deduplication strategy $\mathsf{DS} = (p_1, p_2, p_3, \dots)$

$p_0 = 0$

$$\mathsf{thr} \leftarrow \mathsf{DS.Alg}(\mathsf{F}, \lambda)$$

$\Rightarrow$ server asks for file thr times before saying it already has it

Examples:

- A server that does not defend against the existence-of-file attack: $\mathsf{DS}^{\mathsf{dnd}} = (1, 0, \dots)$
- Threshold chosen uniformly at random: $\mathsf{DS}^{\mathsf{U}} = (\frac{1}{\mathsf{B}}, \frac{1}{\mathsf{B}}, \dots, \frac{1}{\mathsf{B}}, 0, \dots)$

Observations

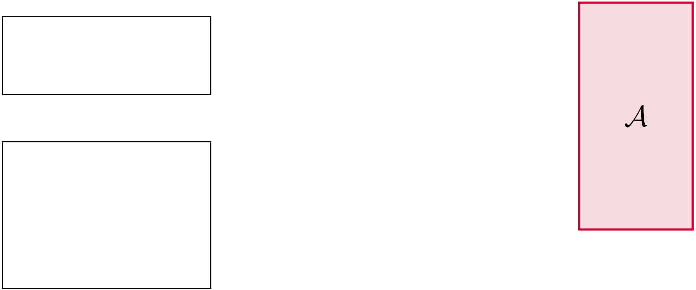
For each strategy, we can define two properties:

- Expected dedup threshold $E = \sum_{i=1}^{\infty} i.p_i$
- Statistical distance $\Delta = \frac{1}{2} \sum_{i=0}^{\infty} |p_i - p_{i+1}|$

E is measure of bandwidth overhead (efficiency).

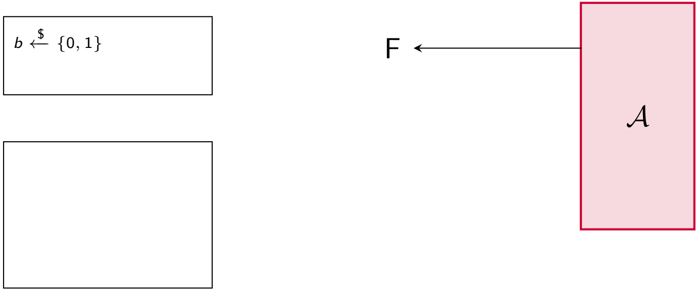
Δ is a relative measure of security of the strategy.

Formalize the IND-EFA game between challenger and an adversary:



Defining Security: Existence-of-file Attack

Formalize the IND-EFA game between challenger and an adversary:



- $\mathcal{A}$ chooses a file F from the filespace. $\mathcal{A}$ attempts to distinguish the two distributions DS and DS*, where DS* is the deduplication strategy probability function shifted one position to the left.

Defining Security: Existence-of-file Attack

Formalize the IND-EFA game between challenger and an adversary:

$b \xleftarrow{\$} \{0,1\}$
thr $\leftarrow$ DS.Alg(F)
ctr $\leftarrow b$

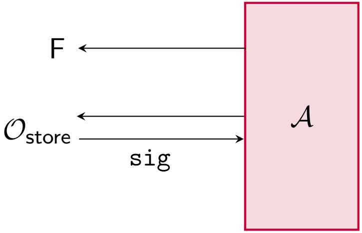
F $\leftarrow$ $\mathcal{A}$

- $\mathcal{A}$ chooses a file F from the filespace. $\mathcal{A}$ attempts to distinguish the two distributions DS and DS*, where DS* is the deduplication strategy probability function shifted one position to the left.
- The challenger invokes strategy algorithm DS.Alg on F and in the $b = 1$ case increments the counter by one to simulate initial storage of F.

Defining Security: Existence-of-file Attack

Formalize the IND-EFA game between challenger and an adversary:

$b \xleftarrow{\$} \{0,1\}$
thr $\leftarrow$ DS.Alg(F)
ctr $\leftarrow b$




```
store():
    ctr ← ctr + 1
    if ctr < thr then
        sig ← 1
    else
        sig ← 0
    return sig
```

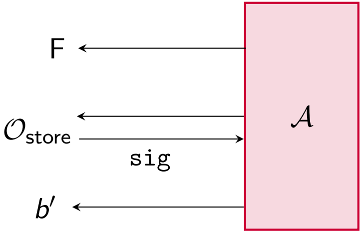
- $\mathcal{A}$ chooses a file F from the filespace. $\mathcal{A}$ attempts to distinguish the two distributions DS and DS*, where DS* is the deduplication strategy probability function shifted one position to the left.
- The challenger invokes strategy algorithm DS.Alg on F and in the $b = 1$ case increments the counter by one to simulate initial storage of F.
- $\mathcal{A}$ has access to a store() oracle which increments the storage counter and responds with the appropriate signal sig.

Defining Security: Existence-of-file Attack

Formalize the IND-EFA game between challenger and an adversary:

$b \xleftarrow{\$} \{0,1\}$
$\textsf{thr} \leftarrow \textsf{DS.Alg(F)}$
$\textsf{ctr} \leftarrow b$

store():
    ctr ← ctr + 1
    if ctr < thr **then**
        sig ← 1
    else
        sig ← 0
    return sig



- $\mathcal{A}$ chooses a file F from the filespace. $\mathcal{A}$ attempts to distinguish the two distributions DS and DS*, where DS* is the deduplication strategy probability function shifted one position to the left.
- The challenger invokes strategy algorithm DS.Alg on F and in the $b = 1$ case increments the counter by one to simulate initial storage of F.
- $\mathcal{A}$ has access to a store() oracle which increments the storage counter and responds with the appropriate signal `sig`.

Defining Security: Existence-of-file Attack

General IND-EFA experiment for deduplication schemes:

$\underline{\mathbf{Exp}^{\text{IND-EFA}}_{\text{DS.Alg},\ \mathcal{A}}(\lambda):}$

$b \xleftarrow{\$} \{0,1\}$
$F \leftarrow \mathcal{A}$
$\text{thr} \leftarrow \text{DS.Alg}(F, \lambda)$
$\text{ctr} \leftarrow b$
$b' \leftarrow \mathcal{A}^{\text{store}}(\lambda)$
return $b' = b$

store():
$\text{ctr} \leftarrow \text{ctr} + 1$
if $\text{ctr} < \text{thr}$ **then**
$\texttt{sig} \leftarrow 1$
else
$\texttt{sig} \leftarrow 0$
return $\texttt{sig}$

$$\mathbf{Adv}^{\text{IND-EFA}}_{\text{DS},\ \mathcal{A}}(\lambda) \stackrel{def}{=} \left| 2 \cdot \mathbf{Pr}\left[\mathbf{Exp}^{\text{IND-EFA}}_{\text{DS},\ \mathcal{A}}(\lambda) = 1\right] - 1 \right|$$

Scheme is IND-EFA Secure if $\mathcal{A}$'s advantage is no better than guessing.

How Effective is a Dedup Strategy that Defends?

For the uniform strategy mentioned earlier
$\mathsf{DS}^{\mathsf{U}} = (\frac{1}{\mathsf{B}}, \frac{1}{\mathsf{B}}, \ldots, \frac{1}{\mathsf{B}}, 0, \ldots)$

How Effective is a Dedup Strategy that Defends?

For the uniform strategy mentioned earlier

$$\mathsf{DS}^{\mathsf{U}} = (\tfrac{1}{\mathsf{B}}, \tfrac{1}{\mathsf{B}}, \ldots, \tfrac{1}{\mathsf{B}}, 0, \ldots)$$
$$E^{\mathsf{U}} = 2 \cdot \tfrac{1}{\mathsf{B}} + \cdots + \mathsf{B} \cdot \tfrac{1}{\mathsf{B}} = \tfrac{\mathsf{B}+1}{2}$$

How Effective is a Dedup Strategy that Defends?

For the uniform strategy mentioned earlier

$$\begin{aligned} \mathsf{DS}^{\mathsf{U}} &= (\tfrac{1}{\mathsf{B}}, \tfrac{1}{\mathsf{B}}, \ldots, \tfrac{1}{\mathsf{B}}, 0, \ldots) \\ E^{\mathsf{U}} &= 2 \cdot \tfrac{1}{\mathsf{B}} + \cdots + \mathsf{B} \cdot \tfrac{1}{\mathsf{B}} = \tfrac{\mathsf{B}+1}{2} \\ \Delta^{\mathsf{U}} &= \tfrac{1}{\mathsf{B}} \end{aligned}$$

How Effective is a Dedup Strategy that Defends?

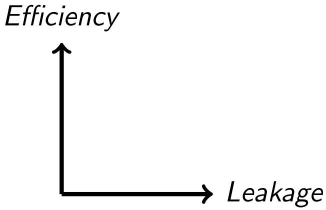
For the uniform strategy mentioned earlier

$$\mathsf{DS}^{\mathsf{U}} = \left(\tfrac{1}{\mathsf{B}}, \tfrac{1}{\mathsf{B}}, \ldots, \tfrac{1}{\mathsf{B}}, 0, \ldots\right)$$

$$E^{\mathsf{U}} = 2 \cdot \tfrac{1}{\mathsf{B}} + \cdots + \mathsf{B} \cdot \tfrac{1}{\mathsf{B}} = \tfrac{\mathsf{B}+1}{2}$$

$$\Delta^{\mathsf{U}} = \tfrac{1}{\mathsf{B}}$$

Tradeoff: bandwidth overhead $\longleftrightarrow$ security level

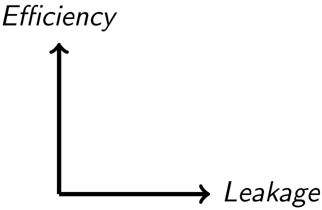


How Effective is a Dedup Strategy that Defends?

For the uniform strategy mentioned earlier

$$\begin{aligned}
\mathrm{DS}^{\mathrm{U}} &= \left(\tfrac{1}{\mathrm{B}}, \tfrac{1}{\mathrm{B}}, \ldots, \tfrac{1}{\mathrm{B}}, 0, \ldots\right) \\
E^{\mathrm{U}} &= 2 \cdot \tfrac{1}{\mathrm{B}} + \cdots + \mathrm{B} \cdot \tfrac{1}{\mathrm{B}} = \tfrac{\mathrm{B}+1}{2} \\
\Delta^{\mathrm{U}} &= \tfrac{1}{\mathrm{B}}
\end{aligned}$$

Tradeoff: bandwidth overhead $\longleftrightarrow$ security level



Natural metric: $E \cdot \Delta$

Theorem

For any deduplication strategy DS with expected threshold E and security level Δ,

$$E^{\mathsf{U}} \cdot \Delta^{\mathsf{U}} \leq E \cdot \Delta.$$

where E^{U} and Δ^{U} are the expected threshold and security level of the uniformly random strategy, respectively.

Proof Outline

Theorem 1

Let $\mathrm{DS} = (p_1, p_2, \ldots, p_{\mathrm{B}}, 0, \ldots)$ *be any deduplication strategy, and let* Δ *and* E *be the corresponding values. Let* π *be a permutation on* $\{1, 2, \ldots, \mathrm{B}\}$ *such that* $\mathrm{DS}' = (p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(\mathrm{B})}, 0, \ldots)$ *is a non-increasing deduplication strategy with corresponding values* Δ' *and* E'*. Then* $\Delta' \leq \Delta$ *and* $E' \leq E$*.*

For any DS, making it non-increasing does not increase $E \cdot \Delta$

Theorem 2

Let $\mathrm{DS}' = (p'_1, p'_2, \ldots, p'_{\mathrm{B}}, 0, \ldots)$ *be a non-increasing deduplication strategy, and let* Δ' *and* E' *be its corresponding values. Then* $1 + \frac{1}{\mathrm{B}} \leq E' \cdot \Delta'$*.*

Given non-increasing strategy, making it 'more uniform' does not increase $E \cdot \Delta$

Future work

- Is this a realistic tradeoff: perhaps $E^s \cdot \Delta$ or other metrics?
- Can we extend this formalism to other attack vectors?
- Extension to other related fields, e.g. cache privacy?

Thank you!
Questions?