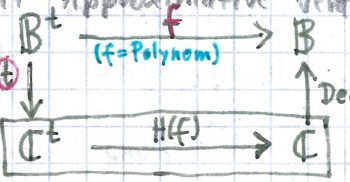
CRYPTO2-Übersicht: VL12[+13]: Homomorphe Verschlüsselung

↳ basieren alle auf Einbauen von Fehlern (LWE)

Geschichte: 1976 Veröffentlichung Diffie-Hellman-Schlüsselaustausch unter dem Namen "ax1x2"
- ☑ sicherer Transport
- ☐ sichere Behandlung?! ↴

Problem 31 Jahre lang offen
- 1978 Rivest, Adleman, Dertouzos "On Data Banks and Privacy Homomorphisms"
  ↳ geben Beispiel-Algorithmen an, aber nur "to illustrate" und "suggestive", "rather weak cryptographically; a [CPA] may break them."
- ↳ 2009 Craig Gentry gelingt in seiner Dissertation die Lösung; (2011 LWE als Grundlage) fand einen wahrscheinlichen Kandidaten für ein FHE-Chiffrierverfahren (2013 Langsames Fehlerwachstum; 2017 Approximative Verfahren umgestiegen).

Überblick & Def.: (Polynom, algebraische NF v. boolsch. Fkt.en)
↳ d.h. ziehe sich erstmal auf Polynome zurück
↳ kann als Vektor auffassen (formal mühsam)



„homomorphe Ausführung von f"

man will eine Fkt. auswerten: $\mathbb{C} \neq \mathbb{B}$ ($\mathbb{C}$=vllt. Vektoren!)

„alles nicht so ganz 100%ig genau"

„abkürzende Schreibw. gewählt; Schlüssel fehlt"

Dabei soll gelten: $f(b_1,\dots,b_t) = Dec(H(f)(Enc(b_1),\dots,Enc(b_t)))$
! verschlüssle bit-weise !

Gängige Schreibweise: $Dec(Eval(f, Enc(b_1),\dots,Enc(b_t))) = f(b_1,\dots,b_t)$

↯↯ Def. homomorphes Kryptosystem: ↯↯

Ein Kryptosystem E heißt homomorph bzgl. einer Menge $F \subseteq \mathbb{B}^\infty$ von Polynomen, wenn gilt:

(↯↯)
- [1] Zum Sicherheitsparameter λ gibt es $Gen(\lambda)$: produziert sk (secret key) $\in \mathbb{B}^k$ [evtl. auch pk (public key)]
- [2] $Enc: \mathbb{B}^k \times \mathbb{B} \to \mathbb{B}^l = \mathbb{C}$ und $Enc^t: \mathbb{B}^k \times \mathbb{B}^t \to \mathbb{C}^t$
- [3] $Dec: \mathbb{B}^k \times \mathbb{B}^l = \mathbb{C} \to \mathbb{B}$ mit $Dec(sk, Enc(sk, x)) = x$.
- [4] $Eval: \mathbb{B}^\infty \to \mathbb{B}^l$ sodass $\forall f \in F: Dec(sk, Eval(f, Enc(sk, b_1), \dots)) = f(b_1, \dots)$
  ⇒ d.h. E ist „korrekt"
- [5] Länge der verschlüsselten Texte ist polynomiell in λ („braucht man") ($\log|\mathbb{C}| = l$; d.h. k!)
  ⇒ d.h. E ist „kompakt" („l könnte ja riesengroß sein, 2 hoch λ") (d.h. $F = \mathbb{B}^\infty$)

(↯↯)

„hier so semi-formal" („geht nachher im Bsp.")

Def. FHE: E heißt Fully Homomorphic Encryption, wenn F alle Polynome enthält.

VL: (Erstes) Bsp.:
- [1] Sicherheitsparameter λ = Länge/Anzahl der Stellen einer Permutation
  Permutation = sk = secret key $\in S_\lambda$ (Symmetrische Gruppe)
- [2] $Enc: S_\lambda \times \mathbb{B} \to \mathbb{B}^\lambda$; $Enc(sk, m) = sk(r \| m)$ wobei r random und $|r| = \lambda - 1$
- [3] $Dec: S_\lambda \times \mathbb{B}^\lambda \to \mathbb{B}$; $Dec(sk, c) = Dec(sk, (c_1,\dots,c_\lambda)) = last\_bit(sk^{-1}(c_1,\dots,c_\lambda))$
- [4] $Eval: \mathbb{B}^\infty \to \mathbb{B}^\lambda$; $Eval(f, c_1, \dots, c_t) = H(f)(c_1,\dots,c_t)$ wobei $c_i = (c_{i,1},\dots,c_{i,\lambda})$
  $= (f(c_{1,1}, c_{2,1}, c_{3,1}, c_{4,1}, \dots, c_{t,1}), f(c_{1,2}, c_{2,2}, c_{3,2}, c_{4,2}, \dots, c_{t,2}), \dots, f(c_{1,\lambda}, c_{2,\lambda}, c_{3,\lambda}, c_{4,\lambda}, \dots, c_{t,\lambda}))$
  $f: \mathbb{B}^t \to \mathbb{B}$
  ← Dec schnappt sich genau 1 von diesen!

↳ (mein) Bsp.: $m = 1 \in \mathbb{B}$; $\lambda = 4$: $sk = \begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix} \in S_4$
$Enc(sk, m) = sk(r\|m) = sk(000\|1) = 0100 = c \in \mathbb{C}$
$Dec(sk, c) = last\_bit(sk^{-1}(0100)) = last\_bit(0001) = 1 = m \in \mathbb{B}$

↳ (mein) Bsp.: $f(x) = x^2$*; Rechenzentrum soll 0010 quadrieren*; verschlüssele 0010 bitweise mit sk wie oben:
$c_1 = sk(101\|0) = 1010$; $c_2 = sk(110\|0) = 0011$;
$c_3 = sk(001\|1) = 1100$; $c_4 = sk(111\|0) = 1011$; Rechenz. rechnet nun:
$Eval(f, 1010, 0011, 1100, 1011) = H(f)(1010, 0011, 1100, 1011)$
$= (f(1011), f(0010), f(1101), f(0101)) = (\dots1, \dots0, \dots1, \dots1)$
$Dec(sk, (\dots1, \dots0, \dots1, \dots1)) = last\_bit(sk^{-1}(\dots)) = \dots0$ ☑ Ergebnis

*mod 2

↯ Schwäche = CPA ↯ ☑ System ist korrekt ☑

Achtung: hier keine Polyname nach ff, sondern viel allgemeiner. Φ=decode; Φ^{-1}=encode g=Generator mod p

[Rivest et al., 1978]: Weitere Bsp.: (1) $U=<\mathbb{Z}_{p-1}, +, ->$ mit $C=<\mathbb{Z}_{p\cdot q}, \cdot, \div>$ und $\Phi^{-1}(x)=g^x \pmod{p\cdot q}$

(2) $U=<\mathbb{Z}_p, \cdot, =>$ mit $\Phi^{-1}(x)=x^e \pmod n$; $(x^e)(y^e)=(xy)^e$; vgl. RSA

(3) $U=<\mathbb{Z}_{p\cdot q}, +, -, \cdot>$ mit $\Phi^{-1}(x)=(x \bmod p, x \bmod q)$; p,q geheim

(4) $U=<\mathbb{Z}, +, -, \cdot>$ mit Darstellg. in Basis n, n geheim

(mein) ↳ Bsp.: (1) Seien $p=11$, $q=3$, $m_1=5\in\mathbb{Z}_{10}$, $m_2=7\in\mathbb{Z}_{10}$ und $g=2$ ein Generator mod 11.

$c_1=\Phi^{-1}(m_1)=2^5 \bmod 11\cdot 3=32$ } $c_3=c_1\cdot c_2=4 \pmod{11\cdot 3}$

$c_2=\Phi^{-1}(m_2)=2^7 \bmod 11\cdot 3=29$ } $\Phi(4)=\log_2(4)=2=m_1+m_2$ ■

↳ $\log_g(g^{m_1}\cdot g^{m_2})=m_1+m_2$ ☑

d.h. $\forall f\in F$: $Dec(Eval(f, Enc(b_1,\dots)))=f(b_1,\dots)$

• **Bootstrapping-Theorem (Craig Gentry):**

Sei E ein F-homomorphes Kryptosystem. Dann kann E zu einem F#E $E_\#$ erweitert werden, wenn in F alle Monome/Polynome bis zu einem gewissen Grad liegen und:

(*) $\forall c_1, c_2\in\Phi$: $[Dec(x,c_1)+Dec(x,c_2)]\in F \wedge [Dec(x,c_1)\cdot Dec(x,c_2)]\in F$

Polynom $f_+^{c_1,c_2}(x)$ Polynom $f_\cdot^{c_1,c_2}(x)$

D.h.: „Wenn in E die Entschlüsselung homomorph ausgeführt werden kann, dann ist E F#E."

Dabei gilt: $E_\#$ ist so sicher wie E, wenn Enc(sk, sk) publiziert werden kann.

↳ **Beweis:** Z.Z.: f_1 und f_2 sind homomorph evaluierbar $\Rightarrow f_1+f_2$ und $f_1\cdot f_2$ sind homomorph evaluierbar

Daraus wiederum lässt sich schließen, dass wir alle Polynome homomorph auswerten können, sprich dass E ein F#E ist. =

(**) Annahme: $Eval(f_1,\dots)=:\sigma_1$ und $Eval(f_2,\dots)=:\sigma_2$ lassen sich korrekt decrypten

Z.Z.: $f_1(\dots)+f_2(\dots)$ und $f_1(\dots)\cdot f_2(\dots)$ lässt sich homomorph evaluieren

Sei $\square=+,\cdot$:

$f_1(\dots)\ \square\ f_2(\dots)$ (**)

$=Dec(sk, Eval(f_1,\dots))\ \square\ Dec(sk, Eval(f_2,\dots))$ | schön & gut, aber noch keine homom. Evaluierg. von $f_1\square f_2$ (da sk vonnöten)

$=Dec(sk,\sigma_1)\ \square\ Dec(sk,\sigma_2)$ ↑Def.

$=f_\square^{\sigma_1,\sigma_2}(sk)$ ↑Definition

$=f_\square^{\sigma_1,\sigma_2}(Dec(sk, Enc(sk, sk)))$ ←Homomorphie

$=Dec(sk, Eval(f_\square^{\sigma_1,\sigma_2}, Enc(sk, sk)))$ ■ ←(*) | (ohne Kenntnis von sk) so lassen sich f_1+f_2 und $f_1\cdot f_2$ homom. evaluieren

Entschlüsseln liefert wie gewünscht

• **„Fully Homomorphic Encryption over the Integers" (Dijk, Gentry et al. 2010):**

(Version 1: „System 1") Sei $sk\in 2\mathbb{N}+1$ eine große ungerade Zahl und $m\in\{0,1\}=\mathbb{B}$ der Klartext.

Verschlüsseln: Wähle $p\in_R\mathbb{Z}$ und $r\in_R\mathbb{Z}$, aber nicht so groß.

$Enc(sk, m)=c=p\cdot sk+2\cdot r+m$ (↯huge blow-up↯) ↳da sk groß sein muss

Entschlüsseln: $Dec(sk, c)=(c \bmod sk) \bmod 2 = m$

Evaluieren/Evaluationsfkt.: $Eval(f, c_1,\dots,c_t)=f(c_1,\dots,c_t)$

↳ fasse Polynom über GF(2) als Polynom über $\mathbb{Z}$ auf

Bsp.: Sei $sk=41\in 2\mathbb{N}+1$ und $m=1$.

$Enc(41, 1)=11\cdot 41+2\cdot 9+1=470$; $Dec(41, 470)=(470\%41)\%2=1$ ✓

$Enc(41, 0)=10\cdot 41+2\cdot 7+0=424$; $Dec(41, 424)=(424\%41)\%2=0$ ✓

Kann NICHT bremen (XOR) $+=$

Sei $f(x_1,x_2)=x_1+x_2$; $f(1,0)=1$; $Eval(f, 470, 424)=470+424=894$

(AND) $\cdot=$

$Dec(41, 894)=(894\%41)\%2=1$ ✓

UE 12: ↯Sei $f(x_1,x_2)=x_1\cdot x_2$; $Dec(41, 470\cdot 424)=0$ funktioniert hier nur durch Zufall↯

↯Problem: Fehler multiplizieren sich: $r_1\cdot r_2\Rightarrow$ Dies ändert Ergebnis modulo 41!↯

(unschöne) Lösungsmöglichkeit: Ciphertext zum Auftraggeber zurückschicken, mit Bitte um ↳schöne Mögl. doch nein UE! Ent- & erneutes Verschlüsseln/„Refreshen", dann kann weiterrechnen

VL 13: ↯auch Bootstrapping hilft nicht, da Dec(·,·) nicht korrekt evaluierbar (Gentry)↯

Forum: ! Komplexität von DEC reduzieren: !

↯ (Version 2: „System 2") Idee: $c \bmod sk=c-sk\cdot\lfloor c/sk\rfloor$ und $(c \bmod sk)\bmod 2=c \bmod 2+\lfloor c/sk\rfloor \bmod 2$ (Addition mod 2)

VL: Könnte man 1/sk veröffentlichen, wäre das „Entschlüsseln" einfach! $sk\equiv 1 \pmod 2$ ↯↯

$\lfloor c/sk\rfloor \leftarrow \mathbb{R}\setminus\mathbb{Z}$ soll jedoch nicht selbst entschlüsseln, aber mir „Tipps" zum Entschlüsseln (z.) geben können. „System 2"

lässt sich aus diesen z. und dem sk berechnen versucht nun also, 1/sk zu verstecken; dummerweise ist Dec(·,·) aber immer noch nicht korrekt↯ evaluierbar