Effektanalysattacker
Del III: Maskering

Anders

20 juli 2022 (14:00)

Sammanfattning

Häri behandlas hur maskering av signaler under exekvering av algoritmer i mjukvara såväl som hårdvara används för att minska informationsläckage och på så sätt göra det svårare för attackerare att gissa sig till skyddsvärda data. Särskilt fokus läggs på effektanalysattacker och hur dessa kan användas för att attackera maskerade implementationer. Exempel på implementering av maskering ges på algoritmisk och arkitekturell nivå.

Innehåll

0 Inledning

Hemlighållande av skyddsvärda data under exekvering av algoritmer är av central betydelse för hemlig kommunikation. En metod för att uppnå det är maskering, i vilken de skyddsvärda data maskeras med en slumpgenererad mask. Detta medför att beräkningar inte sker direkt på de data som anses skyddsvärda och således läcker inte apparaten information om dem.

Maskering kan göras på såväl algoritmnivå, där algoritmen ändras sådant att skyddsvärda data maskeras, som på logiknivå, där vanliga logikelement byts ut mot maskerade logikelement som opererar på maskerade värden.

1 Allmänt om maskering

1.0 Definition

Ett maskeringschema definierar

- ⋆ Hur masker genereras
- ⋆ När värden maskeras
- ⋆ Hur värden maskeras
- ⋆ Hur operationer på maskerade värden utförs
- ⋆ När maskerade värden avmaskeras
- ⋆ Hur maskerade värden avmaskeras

1.1 Syfte

Maskering används för att dölja effektförbrukningen genom att göra den oberoende av signalernas värde i mellanstegen för en kryptografisk algoritm. Detta uppnås genom att utföra beräkningar på maskerade värden i stället för äkta. Maskering innebär att man på algoritmnivå kan hejda att information läcker via sidokanaler. Krypteringsapparaten kan alltså ha en operandberoende effektförbrukning för sina beräkningar men ändå inte läcka information om maskering har implementerats.

Korrekt implementerad maskering ger alltså ett, på en teoretisk nivå, fullständigt skydd mot sidokanalsattacker som förlitar sig på läckage som beror av mellanstegsvärden i en algoritm. Detta är önskvärt vid hantering av skyddsvärda

data som riskerar att röjas för obehöriga och bör ses som en metod av många i hemlighållande av skyddsvärd data.

2 Teori

För varje värde, v, i varje mellansteg under exekveringen av en kryptografisk algoritm maskeras värdet med ett slumpat värde, m, som kallas mask. Det resulterande värdet, $v_m = M(v, m)$, kallas maskerat, och funktionen, M, kallas maskering, och notationen $M(\cdot, m) = M_m(\cdot)$ gäller.

Masken, m, behöver inte vara ett värde utan kan vara många värden. Genom att dela upp sin hemlighet i många delar tvingas attackerare att ta reda på fler värden för att få ut den. I följande exempel kommer masken emellertid vanligen behandlas som ett värde.

Ett maskeringschema beskriver hur maskerna genereras, hur maskeringen sker, hur maskerna ändras under algoritmens gång, hur operationerna på maskerade värden utförs och hur resultatet avmaskeras, $M_m^{-1}(v_m) = v$.

De två vanligaste formerna av maskering är boolesk maskering och aritmetisk maskering.

2.0 Boolesk maskering

Ett känsligt värde, $v \in \mathbb{B}^n$, maskeras med ett slumpvärde, $m \in \mathbb{B}^n$, enligt

$$M_m^{\text{booleks}}(v) = v \oplus m \tag{0}$$

2.0.0 Multiboolesk maskering

Ett ekvivalent maskeringschema som delar upp masken i flera värden kan fås enligt följande: Ett känsligt värde, $x \in \mathbb{B}^n$, delas upp i $d + 1$ olika delar, $x_0, x_1, ..., x_d$, sådana att

$$\bigoplus_{\mu=0}^{d} x_\mu = x \tag{1}$$

En attackerare måste då mäta d signaler för att få ut information om x. Det kallas säkerhet av ordning d.

2.1 Aritmetisk maskering

De två vanligaste formerna av aritmetisk maskering är additiv maskering och multiplikativ maskering. För additiv maskering har vi:

$$M_m^{\text{additiv}}(v) = v + m \mod n \tag{2}$$

där n är ordstorleken för algoritmen i fråga. På samma sätt har vi för multiplikativ maskering:

$$M_m^{\text{multiplikativ}}(v) = v \cdot m \mod n \tag{3}$$

där vi omdelbart noterar de två familjer av fall där $v = 0$ och $m = 0$. Vi har att $M_0^{\text{multiplikativ}}(v) = 0, \quad \forall v$, och på samma sätt $M_m^{\text{multiplikativ}}(0) = 0, \quad \forall m$. Alltså har vi fall där det maskerade värdet, v_m, alltid blir 0 efter maskering respektive fall där v_m förblir 0 oavsett mask.

2.2 Affina ekvationer och homomorfier

En linjär funktion, L, definieras som sådan om den uppfyller

$$L(\lambda x + y) = \lambda L(x) + L(y) \tag{4}$$

för någon definition av multiplikation och addition. En Homomorfi, H, är en strukturbevarande funktion som, givet en operation, $\phi(\alpha, \beta)$, bevarar den och alltså uppfyller

$$H(\phi(\alpha, \beta)) = \phi(H(\alpha), H(\beta)) \tag{5}$$

vilket betyder att linjära funktioner är en typ av homomorfi. För maskering gäller att en homomorfi, H, med avseende på maskeringen, M, av värde, v, med mask, m, blir lätt att implementera eftersom omskrivningen

$$H(M(v, m)) = M(H(v), H(m)) \tag{6}$$

kan göras.

2.2.0 Exempel - multiboolesk maskering för affina funktioner

För affina funktioner är teorin triviell eftersom vi kan utnyttja linjära funktioners homomorfiska egenskaper. För en funktion,

$$f(x, y) = ax + by + c \tag{7}$$

med $a, b, c \in \mathbb{B}^m$ konstanter och $x, y \in \mathbb{B}^m$, har vi

$$f(x, y) = c + \bigoplus_{\mu=0}^{d} f_{lin}(x_\mu, y_\mu) \tag{8}$$

där f_{lin} är den linjära delen av f,

$$f_{lin}(x, y) = ax + by \tag{9}$$

Detta kan enkelt inses genom omskrivningen

$$\bigoplus_{\mu=0}^{d} f_{lin}(x_\mu, y_\mu) = \bigoplus_{\mu=0}^{d} ax_\mu + by_\mu = \tag{10}$$

$$= a \bigoplus_{\mu=0}^{d} x_\mu + b \bigoplus_{\mu=0}^{d} y_\mu = ax + by$$

2.3 Ickelinjära operationer

För en ickelinjär operation blir teorin svårare eftersom de linjära funktionernas homomorfi, inte längre är tillgänglig. För vissa typer av ickelinjära operationer är det lättare att implementera vissa typer av maskering.

2.3.0 Exempel - multiboolesk maskering för multiplikation

För en multiplikation, $z = xy$, får vi $(d+1)^2$ termer på formen $x_\mu y_\nu$, med $\mu, \nu \in \{0, 1, ..., d\}$. Detta är otympligt.

2.3.1 Exempel - multiboolesk maskering för utbyteslådor

En annan, vanligt förekommande, ickelinjär operation är utbyteslådan (engelska: Substitution Box, S-box). Vi har alltså, för en utbyteslåda, S, ingen homomorfi med avseende på $\oplus$:

$$\bigoplus_{\mu=0}^{d} S(x_\mu) \neq S(\bigoplus_{\mu=0}^{d} x_\mu) \tag{11}$$

vilket innebär svårigheter vid implementering.

2.3.2 Exempel - boolesk maskering för uppslagstabeller

En uppslagstabell (engelska: Look-up Table, LUT) fungerar som ett slags enkelt minne, där varje element i värdemängden ligger lagrad för motsvarande adress, alltså element i definitionsmängden. För att gå från en uppslagstabell, f, till en maskerad uppslagstabell, f_m, sådan att

$$f_m(v \oplus m) = f(v) \oplus m \tag{12}$$

krävs en hel del arbete. För varje mask, m_μ, som används måste man löpa igenom alla element i LUTtens definitionsmängd och ändra det lagrade värdet enligt $f(v) \to f(v) \oplus m_\mu$. För N använda masker och LUTtar med definitionsmängd $\mathbb{B}^d$ krävs alltså $N \cdot 2^d$ förändringar för varje LUT.

2.3.3 Exempel - multiplikativ maskering för utbyteslådor

Utbyteslådor grundar sig i operationen att finna en multiplikativ invers för ett element i en galoiskropp, $GF(2^n)$

$$f(x) = x^{-1} \tag{13}$$

Detta är en till synes ickelinjär operation, men den är en homomorfi med avseende på multiplikation. Alltså, för en multiplikativ maskering

$$v_m = v \cdot m \mod n \tag{14}$$

har vi att strukturen bevaras:

$$f(v \cdot m) = (v \cdot m)^{-1} \tag{15}$$

$$(v \cdot m)^{-1} = f(v) \cdot f(m)$$

så att operationen blir lätt att implementera.

2.4 Generering av mask

För att generera en mask används slumptal. Vanligtvis sker genereringen i tre (3) steg:

0. Generera entropi från omvärlden genom att observera till synes oförutsägbara fenomen
1. Efterbehandla genom att verka på slumptal med en diffusiv funktion
2. Utvärdera resultatet

3 Implementering

3.0 Implementering på algoritmnivå

3.0.0 Maskerad multiplikation för boolesk maskering

För en multiplikation av två värden,

$$x \cdot y = z \tag{16}$$

vill vi hitta en metod att maskera. Vi genererar tre masker, ξ, η, ζ, och söker en funktion för maskerad multiplikation, Π, sådan att

$$\Pi(x_\xi, y_\eta, \xi, \eta, \zeta) = x \cdot y \oplus \zeta \tag{17}$$

alltså att multiplikationen ger z maskerat med ζ. För att finna en sådan funktion försöker vi skriva om den i termer av maskerade värden. Vi har att

$$\begin{aligned} x_\xi \cdot y_\eta &= (x \oplus \xi) \cdot (y \oplus \eta) = \\ &= (x \cdot y) \oplus (y \cdot \xi) \oplus (x \cdot \eta) \oplus (\xi \cdot \eta) \end{aligned} \tag{18}$$

och att

$$x_\xi \cdot \eta = (x \cdot \eta) \oplus (\xi \cdot \eta) \tag{19}$$

(och motsvarande för $x \to y, \xi \to \eta, \eta \to \xi$). Vi kan använda de resultaten för att skriva funktionen som

$$\Pi(x_\xi, y_\eta, \xi, \eta, \zeta) = (x_\xi \cdot y_\xi) \oplus (y_\eta \cdot \xi) \oplus (x_\xi \cdot \eta) \oplus (\xi \cdot \eta) \oplus \zeta \tag{20}$$

Vilket är en helt maskerad operation som kan realiseras i fyra additioner och fyra multiplikationer, alltså en del mer avtryck i hårdvara eller en hel del extraoperationer i mjukvara.

3.0.1 Maskerad utbyteslåda för boolesk maskering

Utbyteslådan är det mest utmanande steget i AES-liknande chiffer för booleska maskeringar. Låt oss arbeta i ord om 8 oktetter, dvs 256 byte. Det get oss galoiskroppen $GF(2^8) = GF(256)$. Låt oss vidare representera den som den kvadratiska fortsättningen av $GF(2^4) = GF(16)$, sådant att varje element beskrivs av ett första ordningens polynom, $v_h x + v_l$. Välj ett irreducibelt kroppspolynom, p_0, som alla operationer kommer göras modulus med avseende på. Med den representationen kan inversen av ett element beräknas i endast $GF(16)$-operationer enligt följande (alla operationer modulus $p_0 e$):

$$(v_h x \oplus v_l)^{-1} = v'_h x \oplus v'_l \tag{21}$$

$$v'_h = v_h \cdot w'$$

$$v'_l = (v_h \oplus v_l) \cdot w'$$

$$w' = w^- 1$$

$$w = (v_h^2 \cdot p_0) \oplus (v_h \cdot v_l) \oplus v_l^2$$

Problemet består nu i att utföra denna invers maskerat, så att både argument och resultat förblir maskerade och ingen information kan läcka. På samma sätt som vi representerade vårt värde i två komponenter så maskerar vi med två komponenter och får ett maskerat värde, $(v_h \oplus m_h)x \oplus (v_l \oplus m_l)$. Inversen vi söker blir:

$$((v_h \oplus m_h)x \oplus (v_l \oplus m_l))^{-1} = (v'_h \oplus m'_h)x \oplus (v'_l \oplus m'_l) \tag{22}$$

Med ovan föreskrivna metod (21) kan vi skriva de okända termerna i termer av de kända termerna:

$$v'_h \oplus m'_h = v_h \cdot w' \oplus m'_h \tag{23}$$

$$v'_l \oplus m_l = (v_h \oplus v_l) \cdot w' \oplus m'_l$$

$$w' \oplus m'_w = w^{-1} \oplus m'_w$$

$$w \oplus m_w = (v_h^2 \cdot p_0) \oplus (v_h \cdot v_l) \oplus v_l^2 \oplus m_w$$

Man bör notera att vi explicit räknar ut en invers i $GF(16)$ här, w^{-1}. Detta kan göras enklare genom att representera $GF(16)$ som den kvadratiska utökningen av $GF(4)$. I $GF(4)$ är inversen samma sak som kvadraten, så för ett maskerat värde, $v_m = v \oplus m$, har vi:

$$(x \oplus m)^{-1} = (x \oplus m)^2 = x^2 \oplus m^2 \tag{24}$$

vilket innebär att det är en homomorfi, och således enkelt att implementera.

En sån här implementering är i storleksordningen en faktor 2 till 3 större än en omaskerad implementering med avssende på avtryck på en FPGA som krävs och exekveringstid på en mikroprocessor.

3.1 Implementering på logiknivå

Att maskera på logiknivå innebär att byta ut hårdvaran som algoritmen exekveras på snarare än att förändra algoritmen. Ett logikblock, f, med två ingångar, x, y, och en utgång, z (till exempel en addition) skulle alltså transformeras till ett maskerat logikblock, f_m, med fyra ingånar och två utgångar enligt

$$f(x, y) = z \to f_m(x_\xi, \xi, y_\eta, \eta) = (z_\zeta, \zeta) \tag{25}$$

där ξ, η, ζ är masker. Man delar alltså ingångar och utgångar i maskerade värden och masker.

3.1.0 Maskdelning

Det finns tre (3) huvudsakliga alternativ för hur man ska välja vilken signal som maskeras med vilken mask i en logiknivå-implementering:

0. Unik mask för varje signal: Varje signal har en egen mask och maskerna är parvis oberoende av varandra, vilket också betyder att de maskerade singlerna är parvis oberoende av varandra. Denna metod kräver hög komplexitet vad gäller logikblockens funktionalitet. Antalet masker som krävs är stort. Metoden är således opraktisk för allt utom de minsta byggena.

1. Gruppgemensamma masker: Signalerna delas in i grupper, som var och en tilldelas en mask. Detta minskar antalet masker som behöver genereras. Komplexiteten för logikblok vars operander är maskerade med samma mask är lägre än i fallet att maskerna är unika. Maskerade signaler som passerar från en signalgrupp till en annan, $G \to \Gamma$, måste gå igenom extra logik för att byta sin mask, $m_G \to m_\Gamma$. Att bestämma sig för hur många grupper som skall finnas, och vilka signaler som skall ingå i dessa är inte triviellt.

2. Gemensam mask: Alla signaler maskeras med exakt samma mask. Metoden har lägre komplexitet än de andra. Om två värden beror av varandra kommer även deras maskerade versioner bero av varandra. En nackdel med denna metod är att en förändring av masken innebär att masken måste delas ut till alla signaler. Detta kräver ett stort masknät, vars effektförbrukning kommer vara märkbar.

3.1.1 Maskbytesfrekvens

Masker bör bytas ofta för att minska effektiviteten för högre ordningens DPA. Den uppenbara nackdelen med att byta masker ofta är att de måste genereras, delas ut och appliceras med högre frekvens. Den högsta frekvensen som innebär meningsfullt maskbyte är klockfrekvensen som logiken styrs av.

3.1.2 Maskerad DRPL

Dubbelspårig förladdadningslogik (engelska: Dual Rail Precharge Logic, DRPL) är en logiknivå-implementering som används för att dölja effektförbrukning.

För att maskera den appliceras en mask, m, på varje värde, $v \to v_m$. Vanligen används samma mask för alla signaler, och den byts varje klockcykel. Maskerna delas ut via ett masknät som är stort, och därför läcker en del effektförbrukningsinformation. Ytan som krävs för en maskerad implementering av DRPL är ungefär dubbelt så stor som en vanlig omaskerad CMOS-implementering, och effektförbrukningen är avsevärt större.

3.1.2.0 Exempel - boolesk maskering av DRPL NAND

En NAND-cell är en implementering av en funktion

$$f_{NAND} : \mathbb{B} \times \mathbb{B} \to \mathbb{B} \tag{26}$$

som enklast kan beskrivas med sin sanningstabell:

Argument	Värde och komplement
(0, 0)	(1, 0)
(0, 1)	(1, 0)
(1, 0)	(1, 0)
(1, 1)	(0, 1)

Med andra ord ger den 0 om båda inargumenten är 1 och annars ger den 1. I maskerad DRPL kan den implementeras med hjälp av MAJ-celler. En MAJ-cell med tre argument är en implementering av en funktion

$$f_{MAJ} : \mathbb{B} \times \mathbb{B} \times \mathbb{B} \to \mathbb{B} \tag{27}$$

som även den enklast beskrivs med sin sanningstabell:

Argument	Värde och komplement
(0, 0, 0)	(0, 1)
(0, 0, 1)	(0, 1)
(0, 1, 0)	(0, 1)
(0, 1, 1)	(1, 0)
(1, 0, 0)	(0, 1)
(1, 0, 1)	(1, 0)
(1, 1, 0)	(1, 0)
(1, 1, 1)	(1, 0)

Med andra ord ger funktionen det värde som är vanligast bland argumenten. En maskerad DRPL NAND-cell beskrivs då som en funktion

$$f_{MaskNAND} : \mathbb{B} \times \mathbb{B} \times \mathbb{B} \to \mathbb{B} \tag{28}$$

$$(u, v, m) \mapsto f_{NAND}(u, v) \oplus m$$

$$f_{NAND}(u, v) \oplus m = f_{MAJ}(\overline{u}, \overline{v}, \overline{m})$$

Där det tredje argumentet, m, är masken som används. På precis samma sätt kan komplemtet fås

$$\overline{f_{NAND}(u, v, m)} \oplus m = f_{MAJ}(u, v, m) \tag{29}$$

3.1.2.1 Exempel - boolesk maskering av DRPL d-flip-flop

En d-flip-flop är en vippa med två ingångar, d och clk, och två utgångar, q och $\overline{q}$, sådan att q (och dess komplemen, $\overline{q}$) behåller sitt värde ända tills clk går från låg till hög. När detta inträffar så sätts q till värdet av d (och $\overline{q}$ till $\overline{d}$). För en maskerad d-flip-flop vill vi alltså ha en liknande konstruktion men med en maskerad ingång, d_m, maskerad med en mask, m. Eftersom vi går över flera klockcykler är det inte nödvändigtvis så att m är konstant, utan den kan bero av klockcykel.

Låt d vara det omaskerade värdet som ska latchas maskerat, clk logikens klocka och $m : \mathbb{N} \to \mathbb{B}^n$ vara masken som i varje klockcykel är en bitsträng med längd n. Om vi betraktar apparaten i klockcykeln $\tau \in \mathbb{N}$, så har vi det maskerade värdet $d_{m(\tau)} = d \oplus m(\tau)$. I följande klockcykel, $\tau + 1$, kommer det maskerade värdet vara $d_{m(\tau+1)} = d \oplus m(\tau + 1)$. Om vi nu vill latcha det maskerade värdet, $d_{m(\tau)}$, i tidssteget τ, så ska vi egentligen latcha nästa tidsstegs värde (för det är ju inte förrän efter klockans nästa stigande flank som det kommer kunna användas).

Vi kan alltså bryta ned latchandet i två (2) distinkta steg, först byta mask $m(\tau) \to m(\tau + 1)$, och sen latcha $d_{m(\tau+1)}$. Det första steget kan realiseras med en AND-grind, en OR-grind och en MAJ-cell med tre ingångar:

$$d_{m(\tau+1)} = f_{MAJ}(\overline{m(\tau) \oplus m(\tau + 1)}, \quad f_{OR}(d_{m(\tau)}, m(\tau) \oplus m(\tau + 1)), \quad f_{AND}(\overline{d_{m(\tau)}}, m(\tau) \oplus m(\tau + 1))) \tag{30}$$

Detta värde, $d_{m(\tau+1)}$, kan nu latchas i en helt vanlig d-flip-flop, för att användas redan nästa klockcykel, $\tau + 1$.

För att använda första halvan till att förladda kan man koppla utgångarna till var sin NOR-grind som även har clk som ingång och sedan korsa dem. Alltså, för utgången $q_{m(\tau+1)}$, skulle vi då ha:

$$q_{m(\tau+1)} = f_{NOR}(clk, \overline{q}) \tag{31}$$

vilket skjuter upp output med en halv klockcykel.

4 Speciell maskering

4.0 Homomorf kryptering och förblindning

En intressant situation är den i vilken vi kan verka med samma operator på klartext som på chiffertext och uppnå samma resultat, dvs en homomorfi; det kallas homomorf kryptering. Givet en krypterings- och en dekrypteringsfunktion, E respektive $D = E^{-1}$, samt en operator, Ω, har vi att om ovanstående situation ska gälla måste för varje klartext, d, följande ekvation gälla:

$$\Omega(d) = D(\Omega(E(d))), \quad \forall d \tag{32}$$

Med andra ord ska man lika gärna kunna verka på chiffertext och sen dekryptera som att verka direkt på klartexten. Detta ställer höga krav på såväl operatorn, Ω, som krypteringen och dekrypteringen, E respektive D. Om det emellertid gäller så kan det vara till stor nytta, exempelvis om man vill låta någon annan verka på ens data utan att för den sakens skull avslöja dem (ett praktiskt exempel vore ett sjukhus som vill hålla sina patientdata hemliga men vill anlita någon extern som skall kunna operera på dem utan att se dem).

För asymmetriska algoritmer är vanligtvis aritmetisk maskering ett gott val och kallas förblindning. Givet två agenter, Adelheid och Bosse, som har var sin hemlighet, ett värde, $x \in D_f$, respektive en funktion,

$$f : D_f \to V_f \tag{33}$$

vill Adelheid att Bosse ska räkna ut

$$y = f(x) \in V_f \tag{34}$$

åt henne, men hon vill inte avslöja x, på samma sätt som Bosse vägrar avslöja f. Adelheid skapar sig då en hemlig funktion, E, som är en bijektion,

$$E : D_f \to D_f \tag{35}$$

Om, vilket vi får förmoda när vi sysslar med digital elektronik, $D_f = \mathbb{B}^d$ för något d, så kan E beskrivas som en permutation, $E \in S_d$. Adelheid skickar $E(x)$ till Bosse och får $f(E(x))$ tillbaka. Om Adelheid kan finna en funktion, D, sådan att

$$D(f(E(x))) = y \tag{36}$$

så har hon uppnått sitt mål utan att någon av parterna avslöjat någon hemlighet. Det är inte triviellt, eller ens alltid möjligt, att finna D.

4.0.0 Exempel - förblindning av RSA

Låt x vara klartext, e RSA-exponenten, n RSA-modulus och d den privata nyckeln. En möjlig förblindning är då transformen

$$x \to E(x) = (x \cdot r)^e \tag{37}$$

där r är ett slumptal, $1 \leq r \leq N$, sådant att r och n är relativt prima, alltså att deras största gemensamma delare är 1. Låt oss nu hitta D. Vi har att dekrypteringen av x är på formen

$$f(x) = x^d \mod n \tag{38}$$

Om vi låter f verka på vårt förblindade värde får vi

$$f(E(x)) = (x \cdot r)^{e \cdot d} \mod n \tag{39}$$

$$(x \cdot r)^{e \cdot d} \mod n = x \cdot r \mod n$$

Från vilket vi får den sökta funktionen D

$$D(y) = y \cdot r^{-1} \mod n \tag{40}$$

sådan att

$$D(f(E(x))) = x \tag{41}$$

4.1 Bussmaskering

Bussar för dataöverföring eller adressering har ofta hög kapacitans. Därför är de känsliga mot effektanalysattacker. Ett sätt att skydda sig på är att maskera bussar med något slags strömchiffer.

Ytterligare en åtgärd är att förladda bussar med slumpade värden.

5 Faror och attacker mot maskering

5.0 Kompilatorer

Kompilatorer är smarta nog att effektivisera och optimera program, och gör så utan hänsyn till de explicita instruktioner som beskrivs av källkoden. För att exekvering ska ske enligt ens algoritmer, hur ineffektiva de än må vara, måste kompilatorn hejda sådana optimeringar.

5.1 Hårdvarudesignverktyg

Mjukvaruverktyg som används för design av hårdvara har typiskt en tendens att motverka redundans. Detta är oönskvärt eftersom många maskeringar innebär redundanta operationer så som att uföra en xor två gånger i följd

$$x \oplus y \oplus y = x \tag{42}$$

där funktionen $t \mapsto t \oplus y \oplus y$ kan beskrivas som identitetsavbildningen och således helt plockas bort från logiken.

5.2 Flervärdesberoende

En maskering kan dölja mellanstegsvärden, så att operandberoende effektförbrukning inte avslöjas, men operationer kan bero på mer än en operand så att de fortfarande läcker information.

5.2.0 Exempel - hammingavstånd för boolesk maskering

Operationer som läcker hammingavstånd mellan två mellanstegsvärden, v och w, kommer fortsätta läcka lika mycket om samma mask, m, används för att maskera båda värdena.

$$\begin{aligned} HD(v_m, w_m) &= HW(v_m \oplus w_m) = \\ &= HW(v \oplus w \oplus m \oplus m) = HW(v \oplus w), \quad \forall m \end{aligned} \tag{43}$$

5.3 Samförvaring

Det är olämpligt att lagra ett maskerat värde tillsammans med sin mask i register som läcker hammingavstånd.

5.4 DPA mot maskering

Eftersom alla värden är maskerade under exekveringen av en maskerad algoritm kan DPA inte, på en teoretisk nivå, användas. Emellertid finns möjlighet att utnyttja DPA om slarv förekommer.

5.4.0 Multiplikativ maskering

Multiplikativ maskering uppfyller inte villkoret för oberoende mellan ett värde, v, och dess motsvarande maskerade värde, $v_m = v \cdot m \mod w$, givet en mask, m, och en ordstorlek, w. Detta kommer sig av att i fallet $v = 0$ har vi $v_m = 0 \cdot m = 0, \quad \forall m \in \mathbb{B}^w$. Alltså är multiplikativ maskering sårbart för DPA, särksilt med avseende på nollvärdesmodeller.

5.4.0.0 Exempel - inversion för utbyteslådor

Utbyteslådor kan maskeras multiplikativt. Ett steg i implementationen av utbyteslådor är galoiskroppsinvertering. För ett värde, v, med boolesk maskering, $v_m = v \oplus m$, vill vi alltså göra transformen

$$v_m = v \oplus m \rightarrow (v^{-1})_m = v^{-1} \oplus m \tag{44}$$

Detta kan brytas ned i fem steg. Först genererar vi en till mask, μ, som vi multiplicerar med

$$v_m = v \oplus m \rightarrow (v \oplus m) \cdot \mu \tag{45}$$

Sedan drar vi bort termen med blandade masker genom att använda XOR

$$(v \oplus m) \cdot \mu \rightarrow (v \oplus m) \cdot \mu \oplus m \cdot \mu = v \cdot \mu \tag{46}$$

Resultatet inverteras

$$v \cdot \mu \rightarrow (v \cdot \mu)^{-1} \tag{47}$$

Till detta adderas masken $m \cdot \mu^{-1}$

$$(v \cdot \mu)^{-1} \rightarrow (v \cdot \mu)^{-1} \oplus m \cdot \mu^{-1} \tag{48}$$

Nu återstår bara att multiplicera in μ (kom ihåg att $(v \cdot \mu)^1 = v^{-1} \cdot \mu^{-1}$)

$$(v \cdot \mu)^{-1} \oplus m \cdot \mu^{-1} \rightarrow v^{-1} \oplus m \tag{49}$$

Vilket var vad som söktes. I denna kedja kan vi notera transformerna 47 och 48. I dessa steg verkar vi direkt på värden som enbart är maskerade multiplikativt och de är därför känsliga för DPA med nollvärdesmodeller.

5.4.1 Maskåtervinning

Det kan vara oönskat beräkningstungt eller energiförbrukande att beräkna, dela ut och använda nya masker hela tiden, därför kan det finnas situationer i vilka masker måste återanvändas, både för olika värden och för olika exekveringar av algoritmen.

Det medför risker att använda samma mask till flera maskeringar. Om det ingår en XOR-operation mellan två värden som är maskerade med samma mask på booleskt vis så kommer operationen avmaskera dem och resultatet vara samma som om de inte maskerats överhuvud. I ett sådant fall krävs alltså två olika masker. På samma sätt, om en buss läcker hammingavstånd mellan ett värde och föregående så är det oklokt att skicka två booleskt maskerade värden som använder samma mask i följd på denna buss.

Om samma mask används flera exekveringar i följd, exempelvis om det är beräkningstungt att maskera något steg så kanske man bara vill räkna ut det var tionde exekvering, kommer maskerna inte vara oberoende och DPA kan användas.

I fallet att man kombinerar boolesk och multiplikativ maskering kan problem uppstå om man återanvänder samma mask. Om m och μ från exemplet ovan är samma mask så uppstår problem i och med att masken $m \cdot \mu^{-1}$ som används i flera steg är värdelös eftersom den inte beror av någon slump längre. Ytterligare ett problem är mellanvärdet $(v \oplus m) \cdot \mu = (v \oplus \mu) \cdot \mu$. Problemet här är fallet

$$(v \oplus \mu) \cdot \mu = 0 \tag{50}$$

I fallet $v = 0$ återstår bara $\mu \cdot \mu = 0$ vilket bara har lösningen $\mu = 0$. I fall $v = 1$ fås svaret för både $\mu = 0$ och för $\mu = 1$. Alltså är inte det maskerade värdet oberoende av v utan det finns en statistiskt skev fördelning. Detta leder till att DPA kan användas.

5.4.2 Statistiskt skeva masker

Vi har redan slagit fast att masker skall dras ur en lådformad fördelning. Om maskerna har en statistisk skjuvning så kommer DPA kunna användas.

En metod för en attackerare att införa skevhet i maskerna är genom manipulering av apparaten. Exempelvis kan man försätta apparaten i ett klimat vars temperatur är för hög för att entropigenereringen skall fungera korrekt eller så kan man injicera fel med spänningsspikar (många fler metoder finns givetvis).

En annan metod är att en attackerare mäter några spår och väljer ut en delmängd av dem som motsvarar någon viss delmängd med avseende på maskerna som använts. Ett sätt att utföra detta på är användbart om attackeraren tillåts kryptera samma klartext många gånger. Då kan man med effektmätningar ta reda på när masker genereras och när maskeringar sker. Om det går att få ut någon information om maskerna alls, exempelvis deras hammingvikt, så går det att välja ut delmängd av spår med avseende på maskernas hammingvikt. Således är inte maskerna dragna ur en lådformad fördelning och DPA kan användas.

5.5 Högre ordningars DPA mot maskering

Vad som refereras till som DPA skulle lika gärna kunnat kallas första ordningens DPA. För att dessa termer, första, andra &sv., ska vara meningsfulla krävs det en generalisering av begreppet DPA. Med DPA av ordning n syftar man på DPA

som beror av n olika punkter i effektspåren som beror av olika mellanstegsvärden som använder samma mask.

5.6 Andra ordningens DPA mot maskering

I andra ordningens DPA riktar man in sig mot två (2) olika mellanstegsvärden som är maskerade med samma mask. För att kunna jämföra effektförbrukningens beroende av dessa två olika värden så fungerar inte medelvärdesmetoder så som de beskrivits tidigare eftersom varianserna kan variera. I stället förprocesseras effektspåren så att medelvärdesmetoder kan användas.. Efter förprocesseringen kan metoden fortgå precis som vanlig DPA där de simulerade effektvärdena bygger på någon fysikalisk modell som kombinerar mellanstegsvärdena. De förprocesserade spåren och de kombinerade effektvärdena jämförs och korrelationskoefficient fås.

5.6.0 Mot mjukvaruimplementationer

De mätta spåren transfomeras med en förprocesseringsfunktion, f_{pre}, sådan att $t \mapsto f_{pre}(t) = \tau$. I en mjukvaruimplementation är det typiskt att mellanstegsvärdena man vill mäta inte beräknas under samma klockcykel, och utöver det kan det vara svårt att veta, mer precist, när de beräknas. I stället gissar man sig till ett tidsintervall som förmodligen innehåller beräkning av båda mellanstegsvärdena.

Det finns olika val av f_{pre}. Vissa grundar sig på den absoluta effektskillnaden mellan olika punkter i tiden, andra på kvadraten av summan av effekten vid olika punkter i tiden. En annan metod är att använda ett avskärningsvärde och endast beakta de mätningar som uppfyller $t_\mu > \Lambda$ för någon vald effekt, Λ, och något steg i tiden, μ.

5.6.0.0 Exempel: enbitsfallet för boolesk maskering och hammingvikt-modell

I fallet att de två värdena vi undersöker är enbitsvärden och vi antar att apparaten läcker hamminvikt kan vi enkelt bygga en modell. Vi kombinerar de två värdena, u och v (som är booleskt maskerade med samma mask, m), med en kombineringsfunktion, f_{comb} som är f_{XOR}, till $w = u \oplus v$. Sedan jämför vi de simulerade effektvärdena från den här modellen med de förprocesserade värdena och tar ut korrelationen

$$\rho(HW(u \oplus v), f_{pre}(HW(u_m), HW(v_m))) \tag{51}$$

Låt oss nu undersöka detta resultat för olika val av f_{pre}:

u_m	0	0	1	1	
v_m	0	1	0	1	
$HW(u \oplus v)$	0	1	1	0	
Val av förprocesseringsfunktion					**Korrelationskoefficient**
$HW(u_m) \cdot HW(v_m)$	0	0	0	1	-0.57
$\lvert HW(u_m) - HW(v_m) \rvert$	0	1	1	0	1
$(HW(u_m) + HW(v_m))^2$	0	1	1	4	-0.33
$HW(u_m) + HW(v_m)$	0	1	1	2	0
$HW(u_m) - HW(v_m)$	0	-1	1	0	0

Det bästa valet av f_{pre}, i fallet att apparaten läcker hammingvikt och använder boolesk maskering, är alltså det som tar beloppet av skillnaden mellan de maskerade värdenas hammingvikt. Notera att de linjära valen inte ger någon korrelation alls.

5.6.0.1 Exempel: flerbitsfallet för boolesk maskering och hammingvikt-modell

Om vi utökar föregående exempel till fall där v och u är längre än en bit fås följande korrelationskoefficienter för olika val av förprocesseringsfunktion:

Antal bitar → **Val av förprocesseringsfunktion**	1	2	4	8
$HW(u_m) \cdot HW(v_m)$	-0.58	0.32	-0.17	-0.09
$\lvert HW(u_m) - HW(v_m) \rvert$	1	0.53	0.34	0.24
$(HW(u_m) + HW(v_m))^2$	-0.33	-0.16	0.08	-0.04
$HW(u_m) + HW(v_m)$	0	0	0	0
$HW(u_m) - HW(v_m)$	0	0	0	0

Vi kan dra slutastsen att samma val av förprocesseringsfunktion även är bäst här.

5.6.1 Andra ordningens DPA mot mjukvaruimplementationer med mallattacker

Mallar kan appliceras vid olika skeden av en DPA.

0. Före förprocesseringen. Mallar används till att behandla värdena, exempelvis som ovan $HW(v_m)$ och $HW(u_m)$. Dessa mallbehandlade värden används sedan för förprocessering.
1. Som förprocessering. Mallar används för att välja ut en delmängd av de förprocesserade data för att inducera skevhet i maskfördelningen. Exempelvis kan man ställa ett villkor att hammingvikter måste överskrida något visst avskärningsvärde och förkasta annars.
2. Att applicera mallar efter att spår förprocesserats är i allmänhet ineffektivt.

Under en mallattack matchas mallarna mot spåren. Eftersom värdet på masken, m, som mellanstegsvärdet, v, maskerats med är okänt måste man matcha för alla tänkbara värden på m. Då fås de betingade sannolikheterna för de olika spåren

$$p(t_\mu | k_\nu \wedge m) \tag{52}$$

Ifrån det får vi, på Bayes vis, sannolikheten

$$p(t_\mu | k_\nu) = \sum_m p(t_\mu | k_\nu \wedge m) \cdot p(m) \tag{53}$$

Mallattacker fungerar alltså liknande mot maskerade implementationer som mot omaskerade.

5.6.2 Mot hårdvaruimplementationer

I hårdvaruimplementationer beräknas ofta många värden i samma klockcykel, därför ser förprocesseringen litet annorlunda ut. Effektförbrukningen i en sådan klockcykel beror alltså på flera maskerade värden och deras mask. Vi skiljer mellan två fall och utför förprocesseringen enligt dem.

0. Värdena och deras mask behandlas parallellt. Inga funktioner tar både maskerat värde och mask som argument och följdaktligen finns det ingen koppling mellan modulerna som behandlar de maskerade värdena och modulerna som behandlar masken. I detta fall är förprocesseringen bara summan av effektberoendena. Om en apparat läcker hammingvikt blir förbrukningen proportionell mot $HW(v_m) + HW(m)$. För att introducera icke-linjäritet kan resultatet exempelvis kvadreras. Det finns även icke-linjära förprocesseringsfunktioner.
1. Värdena och deras mask behandlas i samma funktioner. I detta fall krävs oftast ingen förprocessering.

Svårigheterna i DPA-attacker mot hårdvaruimplementationer ligger i att hitta en modell effektförbrukningens beroende av de ingående värdena.

5.7 Maskgenereringsläckage

Maskering förlorar sitt syfte om masken är känd, därför kan maskens skyddsvärde betraktas som så hög som de data som maskeras av den. För att kunna hemlighålla en mask krävs inte bara att den behandlas på ett säkert sätt i logiken utan även att den genereras på ett säkert sätt. Detta ställer krav på slumpgeneratorn. Om en attackerare kan manipulera entropigenereringen eller utföra mätningar på slumpgeneratorn så kan den utifrån det bygga en sannolikhetsfördelning och på så sätt vinna sig fördel i gissandet av mask.

6 Vidare läsning

Lägg till några intressanta artiklar här. Den om drpl verkar ok och den iranska och kanske nån mer får leta litet typ. Tume hade sett nån nyligen på eprint

som jag får titta på så småningom.

A Notation och definitioner

Notationen som används skiljer sig en del från standardnotation inom ingenjörsvärlden och drar mer åt det hållet som används inom teoretisk såväl som matetmatisk fysik. Notationen följer [Fur22a] som läsaren bör konsumerat vid det här laget.

$\mathbb{B}$ används för att beteckna mängden $\{0, 1\}$ som är mycket vanlig inom digital elektronik.

$\oplus$ används för den exklusiva eller-operatorn, som definieras bitvis enligt

$$\begin{aligned} \oplus : \mathbb{B} \times \mathbb{B} &\to \mathbb{B} \\ (\alpha, \beta) &\mapsto \alpha \oplus \beta \\ &= |\alpha - \beta| \end{aligned} \tag{54}$$

Och från den definitionen för längre bitsträngar

$$\begin{aligned} \oplus : \mathbb{B}^d \times \mathbb{B}^d &\to \mathbb{B}^d \\ (\alpha_\mu, \beta_\mu) &\mapsto \alpha_\mu \oplus \beta_\mu \\ &= |\alpha_\mu - \beta_\mu|_\mu \end{aligned} \tag{55}$$

Resultatet blir alltså en bitsträng i vilken varje komponent är resultaten av en $\oplus$ mellan motsvarande komponenter i operanderna.

B Förkunskaper

Läsaren bör ha konsumerat [Fur22a] och [Fur22b] som båda är av förberedande karaktär.

B.0 Naturfilosofi

Artikeln är skriven på ett sätt som gör att den inte går in särskilt mycket i detaljer vad gäller fysiken. Läsaren bör vara någorlunda bekväm inom:

- ⋆ Kretselektronik, se exempelvis [Wes14]
- ⋆ Elektrodynamik, se exempelbis [Fur22c], [Jac62] eller [Nor]

B.1 Matematik

Inga höga krav ställs på matematik för att ta till sig andemeningen i denna artikel, men för att verkligen förstå och kunna gå vidare rekkomenderas att känna sig bekväm med:

⋆ Linjär algebra, se exempelvis [Gus13]

⋆ Multilinjär algebra, se exempelvis [Gre78]

⋆ Reell analys, se exempelvis [Lar05a], [Lar05b], [Kol75], [Spi67] och [Rud66]

⋆ Sannolikhetslära, se exempelvis [Kol56] och [Jay79]

⋆ Fourieranalys, se exempelvis [Fol09]

C Vidare läsning

Härifrån kan en intresserad läsare givetvis fortsätta till [Fur22d].

Är man intresserad av att fördjupa sig i döljning som ämne så kan man ju alltid läsa de böcker och artiklar som bidragit mest till det här kvädet: [MO07], [Eff10], [Pee13], [Sch04], [Ros97], [al99] och [al09].

Vill man ha en djupare förståelse för fysiken bakom effektförbrukning och elektronik är det rimligt att läsa de böcker som använts för skrivandet av denna artikel: [Jac62], [Fur22c], [Nor], [Gri99], [Che89], [Hec15], [Mag05], [Sch95], [Sch00], [Fol08], [Wei05], [Dre65], [Bae92], [Fur22b], [Kra87], [McK93] och [Wes14].

Vill man lära sig mer om matematiken som används inom döljning och attacker mot döljning så är sannolikhetslära och multilinjär algebra de områden som jag anser skulle ge mest lön för ansträngingen. Inom dessa områden rekommenderar jag följande böcker som även använts inom skrivandet av denna artikel: [Kol56], [Jay79], [Gus13], [Gre78], [Dar94] och [Spi70]

D Slumpgeneratorer

En *slumpgenerator* (*eng*: True Random Number Generator, TRNG) samlar entropi från omvärlden genom att utföra mätningar på till synes oförutsägbara fenomen. Dessa kan vara av många olika slag och en djupgående diskussion är olämpligt för den här artikeln[†], men några exempel kan nämnas: Temperatur, accelerometerdata, elektronikjitter, strålning (α, β, γ)[♯] och fysiologiska

[†]Vad som utgör en sann slumpkälla och vilka fenomen som är lämpliga att mäta är kanske inte helt lätt att definiera respektive avgöra. Definitionsmässigt kokar det ned till huruvida det på förhand går att ställa upp en sannolikhetsfördelning sådan att den förutsäger resultatet av en fysikalisk process. Slump, *dolda variabler* och oförutsägbara fysikaliska processer diskuteras utförligt i [Jay79], [Sch95], [Fol08], [Neu32], [Kol56], [Fur22e], [Mag05], [Sch00], [Wei05], [Bae92], [Dre65], [Fol08], [Sha13], [Gri18], [Dir82], [Hen15] och [Wol94].

[♯]Strålning syftar här på emitterade partiklar från sönderfall, typiskt av atomkärnor. Vid övergångar mellan olika tillstånd kan kärnor emittera partiklar. Typiskt är dessa av sex (eller åtta) slag. Protonemission, neutronemission, elektronemission, positronemission, kärnemission och gammaemission. Till dessa har vi även neutrinoemissioner av två olika slag men de är så svåruppmätta att vi inte rimligtvis kan bygga slumpgeneratorer baserade på dem och dessutom sker de alltid i kombination med positronemission eller elektronemission. Elektron- och positronemission kallas för β-sönderfall (ibland skriver man mer specifikt β^- för

rörelser (som puls och leders vinklar). Ju bättre slump desto högre entropi erhålls. Entropin för en slumpad variabel, x, med utfallsrum, $\{x_0, x_1, ..., x_n\}$, och tillhörande sannolikheter, $\{p_0, p_1, ..., p_n\}$, kan definieras såsom (detta värde kallas för *shannonentropi*):

$$\mathscr{S}(x) = -\sum_{\mu=0}^{n} p_\mu \log_2 p_\mu \tag{56}$$

där ett värde av $\mathscr{S} = 1$ innebär fullständig slump, alltså att varje utfall är lika sannolikt.

Under entropiinsamling är det viktigt att slumpgeneratorn inte låter sig styras av yttre påverkan, därför används sensorer och självtester.

D.0 Diffusiva funktioner

En *diffusiv* funktion, f_{diffusiv}, är en endomorfi över någon mängd, M,

$$f_{\text{diffusiv}} : M \to M \tag{57}$$

som inte nödvändigtvis behöver vara surjektiv eller injektiv. De viktigaste egenskaperna för en diffusiv funktion är att det ska vara svårt att hitta tillbakalyftningen (eller ens ett element i den)

$$\begin{aligned} x &= preim_{f_{\text{diffusiv}}}(y) \\ &\in \wp(M) \end{aligned} \tag{58}$$

för något $y \in M$, och att det inte ska finnas något samband mellan funktionens värde och argument för några par av värden och argument. Med andra ord ska man inte kunna ”invertera”[†] funktionen och ”små ändringar” i argument ska inte nödvändigtvis ge ”små ändringar” i värde[♯].

Funktioner av denna typ kallas ibland *envägsfunktioner*.

elektronemission och β^+ för positronemission). Bland kärnemissioner är helium-4 (alltså en kärna med två protoner och två neutroner) så vanlig att den fått ett eget namn, α-sönderfall. Neutronemission och protonemission kallas ibland för n-sönderfall respektive p-sönderfall fast dessa termer är inte så vanliga. När någon av dessa har skett så är kärnan oftast i ett instabilt tillstånd vilket den rättar till genom att avfyra en eller flera fotoner, och det kallas γ-sönderfall. Av dessa är de rimligaste slumpkällorna α-, β- och γ-sönderfall.

[†]Oegentligt med tanke på att f_{diffusiv} inte ens nödvändigtvis är en bijektion, och därmed kanske inte ens har någon invers. Detta uttryck kommer vanligvis av ett missförstånd eller ihopblandning av termerna invers och tillbakalyftning. Tillbakalyftningen av en funktion, $f(x)$, skrivs ibland, något slarvigt, $f^{-1}(x)$, eller, något mindre slarvigt men fortfarande slarvigt, $f^{-1}[x]$.

[♯]Oegentligt med tanke på att vi inte ens definierat vad en liten ändring är. M behöver inte vara utrustad med någon metrik eller norm. I dessa sammanhang är dock $M = \mathbb{B}^d$ där d är något positivt heltal. Vanliga mått på små förändringar på $\mathbb{B}^d$ mellan två element, v och w skulle exempelvis kunna vara $HD(v, w) = HW(v \oplus w)$ eller $|v - w|$ där elementen har en numerisk tolkning, exempelvis så som är vanligt inom digital elektronik, alltså att $v = v_\mu \in \mathbb{B}^d$ har det numeriska värdet $v = \sum_{\mu=0}^{d-1} 2^\mu v_\mu$

D.1 Hashfunktioner

En speciell kategori av funktioner som ligger mycket nära diffusiva funktioner är *hashfunktioner*. En hashfunktion, $f_{\#}$, är en funktion,

$$f_{\#} : \bigcup_{\mu=0}^{\infty} \mathbb{B}^{\mu} \to \mathbb{B}^{d} \tag{59}$$

som uppvisar de egenskaper två centrala egenskaperna som diffusiva funktioner uppvisar. En hashfunktion tar alltså en godtyckligt lång sträng av bitar, $v \in \bigcup_{\mu=0}^{\infty} \mathbb{B}^{\mu}$, och avbildar på en bitsträng av fix längd (d bitar), $f_{\#}(v) \in \mathbb{B}^{d}$.

Denna typ av funktioner är mycket användbara. En sak de kan användas för är att tilldela en godtyckligt stor mängd data ett (nästan) unikt[†] värde av mer hanterlig storlek som (nästan) inte avslöjar någonting om vilka data som orsakade det. Ett annat användningsområde är som efterbehandling av slump.

Hashfunktioner väljs vanligtvis så att de går att implementera effektivt i hårdvara.

D.2 Utvärdering av slump

När entropin är samlad och slumptalet verkats på med en hashfunktion utvärderas resultatet, så att det inte uppträder några mönster. Om allting går rätt till skall tre (3) krav uppfyllas (åtminstone önskar man komma så nära att uppfylla dem som möjligt):

0. Det ska inte gå för en anfallare, givet kunksap om ett tillstånd, att räkna ut nästa tillstånd
1. Det ska inte gå för en anfallare, givet kunksap om ett tillstånd, att räkna ut föregående tillstånd
2. Slumptalen skall följa en lådformad sannolikhetsfördelning i utfallsrummet

Det sista villkoret kan uttryckas som att varje utfall skall vara lika sannolikt som varje annat. Om exempelvis utfallsrummet för x är $\mathbb{B}^{d}$, för något positivt heltal, d, så har vi totalt 2^{d} olika utfall. Sannolikheten för att få utfallet $x_{\mu} \in \mathbb{B}^{d}$ skulle alltså vara $p_{\mu} = \frac{1}{2^{d}} \in \mathbb{R}$. Detta ska givetvis gälla för varje $\mu \in \{0, 1, \ldots, 2^{d} - 1\}$ där μ syftar på en numrering av alla element i $\mathbb{B}^{d}$.

Denna utvärdering sker vanligen genom att genererad slump körs igenom mjukvara som letar efter mönster och mäter entropi. Det går även att behandla

[†]Det är givetvis inte unikt utan, om hashfunktionen är någorlunda vettigt byggd, det finns oändligt många element ur definitionsmängden som motsvarar varje element ur värdemängden, så den är verkligen inte unik. Emellertid kommer man vara tvungen att leta länge om d är ett stort tal, typ hundra eller nåt.

slumpdata med okulär inspektion, för det har visat sig att människan ännu kan tävla med datorer i att se mönster.

Referenser

[Fur22a] Anders Furufors. *Matematisk introduktion*. 2022.

[Fur22b] Anders Furufors. *Effektanalys, Del I: Effektanalysens principer*. 2022.

[Wes14] Westcott och Westcott. *Basic Electronics: Theory and Practice*. Mercury Learning & Information, 2014.

[Fur22c] Anders Furufors. *Elektromagnetiska fält, vågledare och antenner, Del I: Elektrodynamikens principer*. 2022.

[Jac62] David Jackson. *Classical Electrodynamics*. Wiley, 1962.

[Nor] Martin Norgren. *Kompendium i elektromagnetisk fältteori, 2H1250*. Avdelningen för teoretisk elektroteknik, Alfvénlaboratoriet, KTH.

[Gus13] Ivar Gustafsson. *Linjär algebra och numerisk analys*. Institutionen för matematik, Chalmers, 2013.

[Gre78] Werner Greub. *Multilinear Algebra*. Springer Verlag, 1978.

[Lar05a] Arne Persson och Lars-Christer Böiers. *Analys i en variabel*. Studentlitteratur AB, 2005.

[Lar05b] Arne Persson och Lars-Christer Böiers. *Analys i flera variabel*. Studentlitteratur AB, 2005.

[Kol75] Andrej Kolmogorov. *Introductory Real Analysis*. Dover Publications, 1975.

[Spi67] Michel Spivak. *Calculus*. Publish or Perish, 1967.

[Rud66] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.

[Kol56] Andrej Kolmogorov. *Foundations of the Theory of Probability*. Martino Fine Books, 1956.

[Jay79] Edwin Thompson Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 1979.

[Fol09] Gerald Folland. *Fourier Analysis and Its Application*. American Mathematical Society, 2009.

[Fur22d] Anders Furufors. *Effektanalys, Del III: Maskering*. 2022.

[MO07] Mangard och Oswald. *Power Analysis Attacks*. Springer Verlag, 2007.

[Eff10] Rankl och Effing. *Smart Card Handbook*. Fourth. Wiley, 2010.

[Pee13] Eric Peeters. *Advanced DPA Theory and Practice*. Fourth. Springer Verlag, 2013.

[Sch04] Helms och Schmidt och Nebel. "Leakage in CMOS Circuits – An Introduction". I: *Springer Verlag* (2004).

[Ros97] Markus Kuhn och Ross Anderson. "Low Cost Attacks on Tamper Resistant Devices". I: *Springer LNCS* (1997).

[al99] Kocher & al. "Differential Power Analysis". I: *Cryptography Research, Inc* (1999).

[al09] Danger & al. “Overview of Dual Rail with Precharge Logic Styles to Thwart Implementation-Level Attacks on Hardware Cryptoprocessors”. I: *HAL* (2009).

[Gri99] David Griffith. *Introduction to Electrodynamics*. Cambridge University Press, 1999.

[Che89] David Cheng. *Field and Wave Electromagnetics*. Addison-Wesley, 1989.

[Hec15] Eugene Hecht. *Optics*. Pearson, 2015.

[Mag05] Michele Maggiore. *A Modern Introduction to Quantum Field Theory*. Oxford University Press, 2005.

[Sch95] Peskin och Schroeder. *An Introduction to Quantum Field Theory*. CRC Press, 1995.

[Sch00] Franz Schwabl. *Advanced Quantum Mechanics*. Springer Verlag, 2000.

[Fol08] Gerald Folland. *Quantum Field Theory: A Tourist Guide for Mathematicians*. American Mathematical Society, 2008.

[Wei05] Steven Weinberg. *The Quantum Theory of Fields*. Cambridge University Press, 2005.

[Dre65] Bjorken och Drell. *Relativistic Quantum Fields*. McGraw-Hill, 1965.

[Bae92] John Baez. *Introduction to Algebraic and Constructive Quantum Field Theory*. Princeton University Press, 1992.

[Kra87] Kenneth Krane. *Introductory Nuclear Physics*. Third. Wiley, 1987.

[McK93] John McKelvey. *Solid State Physics for Engineering and Materials Science*. Krieger Pub Co, 1993.

[Dar94] Richard Darling. *Differential Forms and Connections*. Cambride University Press, 1994.

[Spi70] Michel Spivak. *A Comprehensive Introduction to Differential Geometry*. Publish or Perish, 1970.

[Neu32] John Neumann. *Mathematische Grundlagen der Quantenmechanik*. Springer Verlag, 1932.

[Fur22e] Anders Furufors. *Kvantinformatik, Del I: Kvantfysikens principer*. 2022.

[Sha13] Ramamurti Shankar. *Principles of Quantum Mechanics*. Springer Verlag, 2013.

[Gri18] David Griffiths. *Introduction to Quantum Mechanics*. Cambridge University Press, 2018.

[Dir82] Paul Dirac. *The Principles of Quantum Mechanics*. Clarendon Press, 1982.

[Hen15] Måns Henningson. *Börja med kvantfysik*. Institutionen för fundamental fysik, Chalmers, 2015.

[Wol94] Haken och Wolf. *The Physics of Atoms and Quanta*. Springer Verlag, 1994.