前瞻議題研究小組專題技術報告

作者：專案研究員 林漢洲

2018 一月重大資安事件新聞

Meltdown 和 Spectre（電腦安全漏洞）

今年一月初，資安研究專家公布他們發現可以說是數十年來少見的電腦 CPU 設計的漏洞，影響的層面和嚴重性都很巨大。連要完全修復，光靠軟體 patch 可能都無法完成。尤其是修復的副作用是 CPU 處理速度可能降至只有原來的七成（因為 CPU 設計原理，近二十年內的個人電腦都受影響）

- 簡單解釋 Spectre 和 Meltdown 原理

  Spectre

  因為 CPU 內的 cache 存取速度比電腦的主記憶體要快。所以，CPU 在執行程式會去預測那些程式碼可能會接下來會被執行，CPU 會先在預取這些程式碼到 CPU cache，以加速執行速度。

  Spectre 的原理，在誘導 CPU 從保護記憶體區載入一個數值。然後，再誘導 CPU 從未保護記憶體區載入另一個已知的數值。如果這個已知的數值載入速度比較快，就可得知這個數值是和從 CPU cache 來的，而且和之前的保護記憶體區載入一個數值有關。

  Meltdown 的原理是利用 CPU 對權限控管的安全漏洞，讓惡意者可以存取任何記憶體保護區，而不受限於權限不足。即使位址空間隔離(address space isolation)和使用準虛擬(paravirtualized)軟體容器所使用的安全規範（security model)，Meltdown 也可突破這些管制。對於雲端公司來說，駭客如果能成功地用 Meltdown 漏洞任意存取他人的雲端資料，這將會嚴重影響雲端科技的接收度與未來發展。

- 如何補救

Meltdown 和 Spectre v1 已有軟體 patch 可供下載（Microsoft Winidows, Apple macOS and Linux). 但比較嚴重的 Spectre v2 則須要由韌體更新來補救。Intel 和 ARM 已發出更新可供下載。雲端公司如 AWS, Google 和 Microsoft 也為他們的雲端伺服器補強這兩個漏洞。VMware 也發出 patch 可供虛擬工具用戶下載補強。

Reference

Spectre (security vulnerability) - Wikipedia
https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)
Meltdown (security vulnerability) - Wikipedia
https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)
Spectre and Meltdown: Cheat sheet
https://www.techrepublic.com/article/spectre-and-meltdown-cheat-sheet/

LINE 的加密協定的介紹與解析

一、前言

針對目前在台灣最多人使用的即時通軟體 LINE, 所使用的加密協定。研究和探討 LINE 利用的演算法，如對稱與非對稱的金鑰加解密及簽章的詳細流程，以及建立用戶端和合法 LINE 伺服器的握手協定，為本報告的主要核心。特別的是，我們著重於點對點的加密(LINE 叫它 Letter Sealing), 一對一的訊息傳送，或是一對多群組。點對點的加密，理論上，因為私鑰是不會離開使用者端， 使用者加密的訊息，傳輸的過程，就算經過任何中繼伺服器(例如：LINE 的伺服器），也是無法被即時通訊提供者解密。這和以前訊息都是明文方式存在伺服器比較起來(雖然在傳送過程有用 TLS 協定加密，到達目的地再解密），安全性大幅加強。

二、LINE 使用的 lightweight handshake 協定 (protocol)

為了保障手機和 LINE 伺服器的傳輸安全，LINE 使用輕量握手協定，以在不影響安全性的前提，減少傳輸的計算量和占用資源，以增加系統的可靠性與效率。當初 What's App 在台灣會被 LINE

迎頭趕上的一個重要原因，大致可歸因為 LINE 的記憶體管理和速度上都優於 What's App。 在使用者經驗方面領先，也是 Secure messenger 除了在安全性強度，也需注意的一個環節。

確保使用者端是連結到正確的 LINE 伺服器，LINE 使用靜態橢圓曲線金鑰(ECC key pairs)。私鑰存在 LINE 伺服器，對應的公鑰則存在 LINE 用戶端的程式。0-RTT (zero round-trip time) 可以達成是因為用戶端已在安裝時儲放了靜態 ECDH 金鑰，所以可節省第一次傳輸的時間。接下來是伺服器和用戶端交換訊息（稱為"握手"），以建立共同的金鑰(transport key), 用來加密互傳的資料。

用戶端 Hello

1. 產用初始的一次性 ECDH 金鑰對 和 16 位元長的用戶 nonce. (nonce 是用來保證握手程序不會被惡意第三者盜取了伺服器用戶間的通訊，用來重覆使用，因為 nonce 是一次性的隨機亂數）

   $(c\_init_{public}, c\_init_{private}) = ECDH_{generate}()$
   $c_{nonce} = random_{secure}()$

2. 產生暫時的傳輸金鑰和初始向量。HKDF (hash-based key derivation function) 是用來產生衍生鑰。HKDF ex 是 extract （萃取，即產生較小的 output）。HKDF exp 是 expand (擴大）。

   $len_{key} = 16$
   $len_{iv} = 16$
   $shared_{temp} = ECDH(c\_init_{private}, static_{public})$
   $MS_{temp} = HKDF_{ex}(c_{public} \,||\, c_{nonce}, shared_{temp})$
   $keyiv_{temp} = HKDF_{exp}(MS_{temp}, \text{“legy temp key”}, len_{key} + len_{iv})$
   $iv_{temp} = keyiv_{temp}[16:31]$

3. 產用用戶端的 ECDH 金鑰對

   $(c_{public}, c_{private}) = ECDH_{generate}()$

4. 用 key_{temp} 和 iv_{temp} 加密這二個 c_{public} 和 app data

   $data_{enc} = ENC(key_{temp}, iv_{temp}, c_{public} || \text{app data})$

5. 送這五個到伺服器[$static_{key\ version}$, c_{public}, c_{nonce}, c_init_{public}, $data_{enc}$] 。$static_{key\ version}$ 是為了讓伺服器能用和用戶端相對應的靜態鑰，所以要指定對的版本。(作者註：LINE 的 Whitepaper 少了加 c_init_{public}

伺服器 Hello

1. 用伺服器端的靜態 ECDH 鑰 和 用戶端初始一次鑰算出暫時傳輸金鑰和初始向量

   $shared_{temp} = ECDH(static_{private}, c\_init_{public})$
   $MS_{temp} = HKDF_{ex}(c_{public}||c_{nonce}, shared_{temp})$
   $keyiv_{temp} = HKDF_{exp}(MS_{temp}, \text{“legy temp key”}, len_{key} + len_{iv})$
   $key_{temp} = keyiv_{temp}[0:15]$
   $iv_{temp} = keyiv_{temp}[16:31]$

2. 用 key_{temp} 解密，並求出 c_{public}

3. 產生一對一次性金鑰對和 16 位元的伺服器 nonce

   $(s_{private}, s_{public}) = ECDH_{generate}()$
   $s_{nonce} = random_{secure}()$

4. 算出 forward-secure(FS) 傳輸鑰和 IV

   $len_{key} = 16$
   $len_{iv} = 16$
   $shared_{FS} = ECDH(s_{private}, c_{public})$
   $MS_{FS} = HKDF_{ex}(s_{nonce} || c_{nonce}, shared_{FS})$
   $keyiv_{FS} = HKDF_{exp}(MS_{FS}, \text{“legy temp key”}, len_{key} + len_{iv})$
   $key_{FS} = keyiv_{FS}[0:15]$
   $iv_{FS} = keyiv_{FS}[16:31]$

5. 算出 handshake state, 並用伺服器靜態簽章私鑰加上簽章

   $state = SHA256(c_{public} || c_{nonce} || s_{public} || s_{nonce})$
   $state_{sign} = ECDSA_{sign}(state, sign_{private})$

6. 用 key_{FS} 和 iv_{FS} 加密 app data

   $Data_{enc} = ENC(key_{FS}, iv_{FS}, app\ data)$

7. 送到用戶端

   $[s_{public}, s_{nonce}, state_{sign}, data_{enc}]$

用戶端最後的階段

1. 檢查 handshake 簽章是否一致，

   $valid = ECDSA_{verfiy}(state_{sign}, sign_{public})$

2. 算出 key_{FS} 和 iv_{FS}

   $MS_{FS} = HKDF_{ex}(c_{nonce} || s_{nonce}, shared_{FS})$
   $keyiv_{FS} = HKDF_{exp}(MS_{FS}, \text{“legy fs key”}, len_{key} + len_{iv})$
   $key_{FS} = keyiv_{FS}[0:15]$
   $iv_{FS} = keyiv_{FS}[16:31]$

3. 用 key_{FS} 和 iv_{FS} 加密之後的 application data

握手程序完成後，用戶和伺服器端都用相同的 forward-secure 對稱金鑰 key_{FS}.

三、LINE 的點對點加解密分析

LINE 一對一訊息加密分析

LINE 將預設所有使用者為使用 點對點的加密機制 ，叫作 Letter Sealing.

用戶端一開始要先產生一對 Letter Sealing ECDH 金鑰，存放在用戶端。

並上傳公鑰至 LINE 伺服器, 而且會從 LINE 伺服器傳回一個唯一的金鑰代碼。

如何一對一傳送訊息

首先想發送訊息的使用者要先取得收件人的公鑰。共同對稱鑰則是由下面公式：

Shared Secret
= ECDH $_{curve25519}$ (key $_{user1\text{-}private}$, key $_{user2\text{-}public}$)
= ECDH $_{curve25519}$ (key $_{user2\text{-}private}$, key $_{user1\text{-}public}$)

利用 Shared Secret, 衍生出 用來對訊息加密的 Key $_{encrypt}$ 和 IV

Key $_{encrypt}$ = SHA256(Shared Secret || salt || “Key”)
IV $_{pre}$ = SHA256(Shared Secret || salt || “IV”)
IV $_{encrypt}$ = IV $_{pre}$[0:15] ⊕ IV $_{pre}$ [16:31]

用 256 位元 AES-CBC 加密後的訊息為密文 C

C = AESCBC(Key $_{encrypt}$, IV $_{encrypt}$, M)

為了要證明訊息未被他人不當更改，算出簽章 MAC

MAC $_{plain}$ = SHA256(C)
MAC $_{enc}$ = AESECB(Key $_{encrypt}$, MAC $_{plain}$[0:15]⊕MAC $_{plain}$[16:31])

[version, content type, salt, C, MAC $_{enc}$, sender key ID, recipient key ID]

收信者收到後，利用 Shared Secret, Key $_{encrypt}$(利用上面 sender 用的公式算出來), IV 可以解密。再算出 MAC，如果和 sender 送來的 MAC 吻合, LINE 就會顯示出解密後明文。如果不合，就表示該訊息已被”污染”，將不會顯示出來給用戶。

四、LINE 一對多訊息加密分析

在 LINE 群組聊天時，需要由群組的發起人 (或是想送第一個訊息給群組的人）產生一把共享的群組金鑰 Sharedkey*group*，然後，群組其它成員也會收到這把共享的群組金鑰。

就像一對一的加密，其實這把群組金鑰 Sharedkey*group*，就像是一對一的私鑰，只是在群組裏，大家都用同一把私鑰來加解密群組訊息。

要讓這把 Sharedkey*group* 可以安全地送到群組其它成員，作法如下：

首先，先收集其它成員的公鑰，舉例來說，Bob. (Alice 是送訊息者)

$KEY_{for\text{-}bob} = ECDH_{cuver25519}(KEY_{alice\text{-}private}, KEY_{bob\text{-}public})$

把 Sharedkey*group*

用 $KEY_{for\text{-}bob}$ 加密後放在 messaging server。

當 Bob 要送訊息時，Bob 向 server 要 Encrypted($KEY_{for\text{-}bob}$, Sharedkey*group*)

然後 Bob 可以用這把 $KEY_{for\text{-}bob}$ 來解密，取得 Sharedkey*group*

$KEY_{for\text{-}bob} = ECDH_{cuver25519}(KEY_{bob\text{-}private}, KEY_{alice\text{-}public})$

當群組人員有異動（新加入或離開）時，新的 Sharedkey*group* 要重新產生。以確保離開的人員不能再加解密這個群組之後的訊息。

以上說明，如何把從 sender 送到群組其它成員 Sharedkey*group*

Bob 拿到 Sharedkey*group* 後，把它存放在 local cache, 所以以後不用再向 server 取。

$Shared\ Secert_{group} = ECDH_{curve25519}(SharedKey_{group}, Key_{sender\text{-}public})$

$Key_{encrypt} = SHA256(Shared\ Secret_{group} \| salt \| \text{“Key”})$

$IV_{pre} = SHA256(Shared\ Secret_{group} \| salt \| \text{“IV”})$

$IV_{encrypt} = IV_{pre}[0:15] \oplus IV_{pre}[16:31]$

附註，這 $KEY_{for\text{-}bob}$ 的算法是根據 ECDH (橢圓曲線迪菲-赫爾曼金鑰交換, Elliptic-curve_Diffie–Hellman)

Shared Secret

= ECDH $_{curve25519}$(KEY $_{user1\text{-}private}$, KEY $_{user2\text{-}public}$)

= ECDH $_{curve25519}$(KEY $_{user2\text{-}private}$, KEY $_{user1\text{-}public}$)

五，結語

LINE 採用點對點加密，並預設所有使用者都自動採用此模式，可大幅改善其訊息傳遞的安全性。其前提是 LINE 保證所有用戶端的私鑰被安全保管存放在安全的地點，只在用戶端產生且不會離開用戶端，因此用戶間的通訊，理論上無法被 LINE 公司或員工所解密或監聽。WhatsApp 所公佈的點對點加密方法和群組交換金鑰和 LINE 的作法基本上是十分相似。未來我們會對這些主要即時通訊軟體的通信協定作更詳細深入的探討和研究。

參考資料

[1] LINE Encryptino Overview (Technical Whitepaper) ver. 1.0, September, 2016
https://scdn.line-apps.com/stf/linecorp/en/csr/line-encryption-whitepaper-ver1.0.pdf