時間存儲折中攻擊前沿簡介

王文浩[1,2]，林東岱[1]
[1] 中國科學院信息工程研究所 信息安全國家重點實驗室
[2] 中國科學院大學
{wangwenhao, ddlin}@iie.ac.cn

摘要

時間存儲折中攻擊方法是一種通用的求解函數原象的方法，廣泛應用於區塊編碼器演算法、流密碼演算法和雜湊演算法的密碼分析以及口令認證系統破解等，對區塊編碼器演算法和流密碼演算法的設計也具有參考意義。本文系統介紹了時間存儲折中攻擊方法的攻擊步驟，總結了該方法在區塊編碼器演算法和流密碼演算法分析以及口令破解中的實際應用。

關鍵詞：時間存儲折中攻擊方法、對稱密碼分析、單向函數

壹、單向函數和公開金鑰密碼學

在電腦科學領域，單向函數是指一類容易計算但難以對任意隨機象值求解原象的函數，這裡的「容易」和「困難」是通過在計算複雜度理論模型下是否存在多項式時間演算法來區分的。

單向函數在電腦科學的許多領域，如密碼學、身份認證中等都有著重要的地位，單向函數是否存在仍然是電腦科學中的一個開放性問題。事實上，如果單向函數存在，將證明複雜性類 P/NP 問題中，P 不等於 NP；與之相對，P 不等於 NP 的假設並不能直接推出單向函數的存在。複雜性類 P/NP 問題也是電腦科學中一個懸而未決的重要問題。

公開金鑰密碼演算法常常被用於在開放的網路如網際網路上進行安全的電子通信，它包括一對數學上相關的公開金鑰和私密金鑰，其中公開金鑰是公開的，而私密金鑰是秘密保存的，已知公開金鑰時得到對應的私密金鑰在計算上是不可行的。公開金鑰密碼系統的構建很大程度上是基於(假設存在的)陷門單向函數，給定某個陷門單向函數的實例 F，擁有陷門的人可以正向和逆向計算函數 F，未擁有陷門的人只能正向計算而難以逆向計算函數 F。公開金鑰密碼系統中公開金鑰包含了該函數的描述，私密金鑰對應於陷門。正向計算可用於加密和簽名的驗證，逆向計算可用於解密和簽名的生成。

至今密碼學家還未能從理論上證明這樣的陷門單向函數是存在的，現在所有實際應用的公開金鑰密碼系統都基於某些密碼學假設。如果找到一個演算法能夠在不知道陷門的情況下容易地計算 F 的原象，那麼所有基於函數 F 構建的公開金鑰密碼系統都是不安全的。

與單向函數類似的一個概念是單向雜湊函數。雜湊函數是把可變輸入長度的串(原象)轉換成固定輸出長度的串(雜湊值)的一類函數，雜湊函數是典型的多到一的函數。單向雜湊函數是在一個方向上工作的雜湊函數，一般要求滿足單向性和抗碰撞性。

雜湊函數是許多密碼協定的結構模組，在電腦科學和密碼學中有著廣泛的應用。現在使用較多的雜湊函數演算法包括 MD5[1]、SHA1[12]等。例如，在網站論壇系統、作業系統和資料庫系統等基於口令認證的系統中，網站伺服器或作業系統多採用存儲使用者帳號和口令的雜湊值的方法加密存儲。使用者登陸時只需驗證使用者帳號和口令的雜湊值是否與存儲的雜湊值相同，即可驗證使用者帳號和口令的正確性。如果攻擊者利用某些手段獲取了加密存儲檔，他也不能直接獲得使用者的帳號和口令。

貳、單向函數和對稱密碼學

與公開金鑰密碼學不同，對稱密碼學研究在擁有共用金鑰的通信雙方之間建立安全通信通道。通信雙方秘密保存的共用金鑰記為 k，發送方使用加密演算法 E 和金鑰 k 加密需要發送的消息 m 得到密文 $c = E(k, m)$，並把 c 發送到接收方；接收方使用解密演算法 D 和共用金鑰 k 解密密文恢復出明文 $m = D(k, c)$。對稱密碼學中通信雙方使用同樣的金鑰 k 進行加密和解密，加密和解密演算法是公開的，任何知道金鑰 k 的人可以解密獲得明文消息，共用金鑰 k 必須秘密保存。

對稱密碼演算法分為兩類：一次只對明文的單個比特(有時是位元組)運算的演算法稱為序列密碼演算法(或流密碼演算法)，一次對明文的一組比特(稱作一個分組)運算的演算法稱為區塊編碼器演算法。

2.1 單向函數和區塊編碼器演算法

一個區塊編碼器演算法首先是一個對稱密碼演算法，其中 M=C=$\{0,1\}^n$，金鑰空間為 K=$\{0,1\}^r$，記作

$$E: \{0,1\}^r \times \{0,1\}^n \rightarrow \{0,1\}^n, (k, m) \rightarrow E(k, m)$$

也就是說，使用一個 r 比特的金鑰 k，加密演算法 E 對固定長度 n 的明文分組加密，得到固定長度 n 的密文分組 $c = E(k, m)$，n 稱作對稱密碼演算法 E 的分組長度。固定一個金鑰 k，加密函數 $E_k: m \rightarrow E(k, m)$是$\{0,1\}^n$上的一個置換。

區塊編碼器演算法的應用領域從電子郵件加密、網路通信加密到銀行交易轉帳等非常廣泛，典型的區塊編碼器演算法如 DES 和 AES[10]是美國政府核定的標準密碼演算法。20 世紀 80 年代，Unix 作業系統使用 DES 演算法以使用者口令為金鑰組明文進行加密，所得結果存儲於本地磁片；Windows 作業系統使用了基於 MD4 或 DES 的加密方式。

固定對稱密碼演算法加密的明文 m 和在金鑰 k 下的加密函數E_k，定義函數 $c = f(k) := E_k(m)$。如果可以求解函數 f 的原象，可以在選擇明文或已知明文情景下恢復區塊編碼器演算法的金鑰 k。

為了在各種應用場合使用區塊編碼器，美國在 FIPS PUS 74 和 81 中定義了區塊編碼器的 4 種工作模式：電話本(ECB)模式、密碼分組連結(CBC)模式、密碼回饋(CFB)模式、輸出回饋(OFB)模式。

2005 年，文獻[17][18]指出為了攻擊各類工作模式下的區塊編碼器，定義一個單向函數

$$f: K \times V \to C,$$

其中 K 代表金鑰空間，V 代表各種工作模式下區塊編碼器演算法已知的輸入空間，C 代表各種工作模式下區塊編碼器演算法的輸出空間。按照表一指定的 K、V 和 C 定義單向函數 f，在適當的攻擊情景(如選擇明文或者已知明文)下，通過求解函數 f 的原象即可恢復出區塊編碼器演算法的金鑰。

表一 各類工作模式下的區塊編碼器與單向函數

工作模式	*K*	*V*	*C*
ECB，CTR	金鑰	-	一個密文分組
OFB，CBC，CFB	金鑰	*IV*	密文分組
TBC	金鑰	亂數	密文分組
OCB	金鑰	亂數	密文分組和消息鑒別碼
CMC，EME	金鑰	*tweak*	密文分組

2.2 單向函數和流密碼演算法

記金鑰集合為 Z，明文集合為 M，使用金鑰流 $z := z_1 z_2 z_3 \ldots$，一個流密碼機制

$$E^*: Z^* \times M^* \to C^*, E^*(z, m) = c := c_1 c_2 c_3 \ldots$$

對明文資料流程 $m := m_1 m_2 m_3 \ldots$ 加密得到密文資料流程 $c = c_1 c_2 c_3 \ldots$，映射

$$E: Z \times M \to C$$

使用金鑰z_i對明文m_i加密得到密文

$$c_i = E_{z_i}(m_i) = E(z_i, m_i), i = 1, 2, \ldots$$

一般情況下，z_i、m_i 和 c_i $(i = 1, 2, \ldots)$ 是二進位數位，並且

$$E(z_i, m_i) = z_i \oplus m_i$$

通信雙方約定同一個金鑰 k，使用流密碼演算法生成輸出金鑰流 z。為了防止同一個金鑰組不同的明文加密可能產生的攻擊，流密碼演算法需要另一個輸入初始向量 iv，iv 是公開的以使得接收方能夠解密消息。通常流密碼演算法的輸出金鑰流是由不斷更新的

內部狀態生成的，金鑰 k 和初始向量 iv 填充作為初始內部狀態。如果攻擊者能夠恢復出金鑰 k 或者某個時刻的內部狀態，就能夠解密(該時刻以後的)通信內容。

文獻[5][6][7][8]提出通過求解函數

$$f:\left(\text{內部狀態}\right)\rightarrow\ \left(\text{金鑰流分段}\right)$$

的原象可以恢復流密碼某個時刻的內部狀態。通過攻擊這樣的單向函數進而恢復內部狀態的方法有一個優點：例如，假定內部狀態長度為 6，如果攻擊時獲得t_0時刻開始的金鑰流 001101011，記t_0開始 4 次內部狀態分別為$state_i\ (i\ =\ 0,1,2,3)$，只需通過求解函數 f 的原象獲取其中一個時刻的內部狀態，則攻擊者可以獲得該時刻後的輸出金鑰流(圖一)。

文獻[11][19]提出通過求解函數

$$f^*:\ (K,IV)\ \rightarrow\ (\text{金鑰流分段})$$

的原象可以恢復出流密碼的金鑰。如果獲得消息發送方使用同一金鑰 k 和 m 個不同 $iv_i(i\ =\ 0\ldots\ m-1)$生成的 m 個金鑰流，記作 $seg_i\ (i\ =\ 0\ldots\ m-1)$1，只需求解函數$f^*$任意一段金鑰流的原象，攻擊者可以解密獲得使用者的金鑰 K(圖二)。

$f(state_0)=001101$
$f(state_1)=011010$
$f(state_2)=110101$
$f(state_3)=101011$

圖一 $f:\left(\text{內部狀態}\right)\rightarrow\left(\text{金鑰流分段}\right)$

$f^*(k,iv_0)=seg_0$
$f^*(k,iv_1)=seg_1$
$f^*(k,iv_2)=seg_2$
$f^*(k,iv_3)=seg_3$

圖二 $f^*:(K,IV)\rightarrow(\text{金鑰流分段})$

參、求解函數原象的時間存儲折中攻擊方法

本文討論如何從應用的角度探討如何快速地求解函數的原象，並介紹一種通用的求解函數原象的方法，稱作時間存儲折中(TMTO)攻擊方法，它把函數看成黑盒子，不關心函數的內部結構。

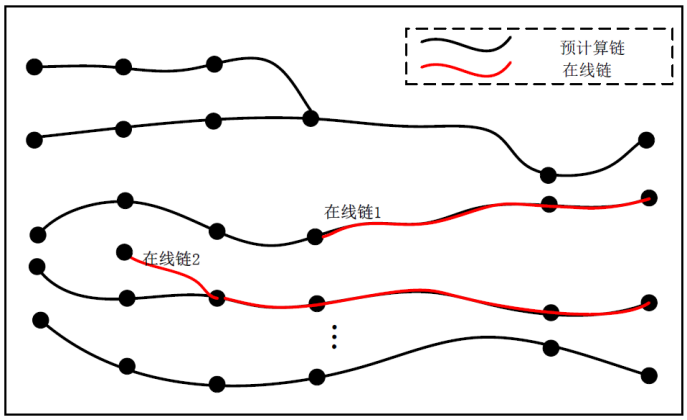
記需要求解原象的函數為f，給定 $y^*\ =\ f(x^*)$，記單向函數f的搜索空間大小為N。自然地存在兩個方法求滿足條件的x使得$f(x^*)\ =\ y^*$。

1）窮舉攻擊。遍歷定義域空間的所有x，檢驗$f(x)$是否等於y^*。

2）查表攻擊。預計算定義域空間的所有x及$f(x)$，按照$f(x)$的大小順序存儲至預計算表。給定y^*，在預計算表中檢索y^*即可求得滿足條件的x^*。

1 這裡要求是同壹時刻開始的多個密鑰流

窮舉攻擊不需要額外的存儲空間，耗費的時間接近於搜索整個定義域空間的時間。查表攻擊耗時極短，缺點是需要存儲所有的明密文對，而且預計算時間接近於搜索整個定義域空間的時間；幸運的是，預計算過程只需實施一次。



圖三 時間存儲折中攻擊方法

Hellman 在文獻[12]中提出針對區塊編碼器 DES 的時間存儲折中攻擊方法，可以做到存儲空間和線上攻擊時間之間的平衡，稱作 Hellman 時間存儲折中攻擊方法或簡稱 Hellman 方法。

攻擊過程分為兩個階段。

預計算階段。攻擊者從隨機選取的值開始，通過反覆運算計算生成多條等長的鏈。預計算鏈生成以後，只有鏈的頭節點和尾節點對按照終點的大小順序存儲至預計算表，所有中間節點都被忽略掉。預計算階段只執行一次，其時間大致相當於窮舉攻擊的時間。

線上攻擊階段。攻擊者從給定的象值y^*開始通過反覆運算計算生成線上鏈。每反覆運算生成線上鏈中的一個值，查找該值是否是某條預計算鏈的尾節點。如果反覆運算值和某條預計算鏈的尾節點匹配，則x^*可能出現在該預計算鏈中，需要該條預計算鏈的頭節點開始重新生成預計算鏈，並找到x^*的一個候選 $\hat{x}$ (圖三：線上鏈 1)。由於鏈可能交叉或者存在自環，候選值 $\hat{x}$ 可能不滿足$f(\hat{x}) = y^*$ (圖三：線上鏈 2)，攻擊者需繼續生成線上鏈，直至找到滿足條件的x^*或者線上鏈達到鏈的固定長度，此時攻擊失敗。

為了保證線上攻擊階段的成功率，預計算階段需要預計算多張表，每張預計算表包含的鏈的個數和鏈的長度之間也滿足一定的關係。

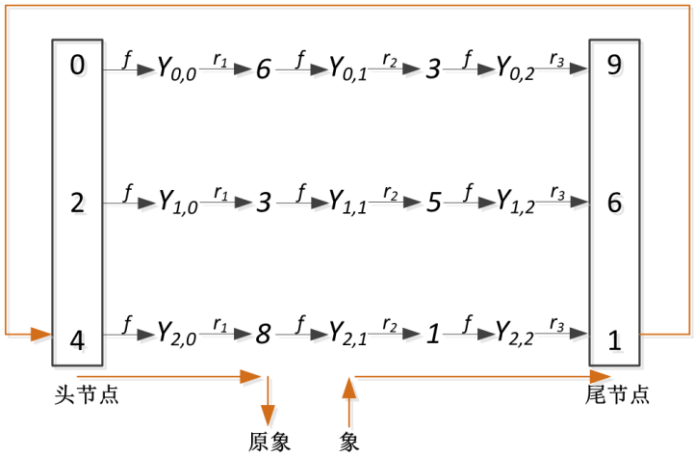
一般記預計算表的大小為M，線上攻擊的時間為T，預計算時間為P，則時間存儲折中攻擊方法滿足 $TM^2 = cN^2$，其中c是一個常數，稱作折中係數。折中係數越小，則時間存儲折中攻擊方法的效率越高。預計算表是反覆運算生成的，不一定能夠完全覆蓋函式定義域的全部空間，因此時間存儲折中攻擊方法是一個概率性的演算法，記線上攻擊階段的成功概率為p，成功率大致隨預計算表數量的增多而增大。如果線上攻擊階段找到x^* 的一個候選 $\hat{x}$，但是$f(\hat{x}) \neq y^*$，則稱發生了一個誤報。

線上攻擊時間由三部分組成：線上鏈的生成時間、預計算表的查找時間、重新生成預計算鏈的時間，其中重新生成預計算鏈的目的是檢測和發現誤報，與誤報發生的位置以及誤報發生的概率有關。線上鏈的生成時間和重新生成預計算鏈的時間，以單向函數 f 的一次反覆運算為單位；反覆運算值的查找時間以磁片的一次查找為單位。二者的單位不統一，與具體的磁片性能和函數執行速度有關。

線上攻擊階段時每次反覆運算生成線上鏈的一個值，都需要在預計算表中查找它是否是某條預計算鏈的尾節點，因此時間存儲折中攻擊方法需要執行多次磁片讀取操作。為了減少磁片讀取次數，文獻[28][30]提出了一種 Hellman 攻擊方法的改進方案，稱作區分點表方法。

與 Hellman 方法不同，區分點表方法採用了不定長的預計算鏈。預計算階段，每條預計算鏈不斷反覆運算計算直到滿足某個預定義條件(例如前k個比特為 0，稱之為區分點)的點出現時才停止。線上攻擊階段，從給定的y^*開始通過反覆運算計算生成線上鏈，只有當線上鏈中出現區分點時才需要對預計算表執行查找操作。預計算階段一般指定預計算鏈的最大長度，當預計算鏈超過最大長度仍沒有到達區分點時，停止生成並扔掉這條預計算鏈。

文獻[9]提出扔掉預計算表中終點相同的鏈，即預計算表由終點各不相同的預計算鏈構成，稱作完美表。使用完美表時，線上鏈的每個反覆運算值最多與一條預計算鏈相交，因此減少了誤報對線上攻擊時間的影響。但是一部分預計算鏈在計算完成後被扔掉，增加了預計算階段的時間。



圖四 彩虹表攻擊方法

文獻[26]中首次提出彩虹表攻擊方法(見圖四)。與 Hellman 表不同，彩虹表預計算表的每一列使用了不同的歸約函數，兩條鏈相交併合並僅當它們在同一列的值相同。彩虹

表不僅提高了每張預計算表的覆蓋率，允許使用更長的預計算鏈，而且減少了誤報對線上攻擊時間的影響。

按照之前的討論，如果線上鏈和預計算鏈合併而產生了一個報警，只有從頭節點開始重新計算預計算鏈，才能區分是成功找到一個原象還是發生了一個誤報。檢查點技術是一項用於排除誤報的技術[3][1]。預計算階段除存儲預計算鏈的頭節點和尾節點外，預計算鏈中間節點的某些資訊也被存儲起來；線上攻擊階段，預計算鏈中間節點的資訊用於在不重新生成預計算鏈的情況下檢測誤報，因而大大減少誤報對線上攻擊時間的影響。

時間存儲折中攻擊方法的其它優化技術還包括變種區分點表[14]、指紋表[1]、尾節點截斷技術[4][8]等。

時間存儲折中攻擊方法最初提出用來攻擊區塊編碼器 DES[20]，文獻[7]提出使用時間存儲折中攻擊方法攻擊流密碼演算法，針對的單向函數是由流密碼內部狀態到金鑰流之間的映射，文獻[8]使用時間存儲折中攻擊方法有效地攻擊了流密碼演算法 A5/1。文獻[19]對時間存儲折中攻擊方案在各種場景下的流密碼、區塊編碼器、雜湊函數等做了全面總結。

彩虹表提出以來，時間存儲折中攻擊方法在實際應用中尤其是在 Windows 登錄口令破解中得到廣泛應用。世界各地的破解愛好者參與到彩虹表工程中，專門開發了 CPU 版本、GPU 並行加速版本和基於 FPGA 實現的彩虹表破解工具，如 Ophcrack、RainbowCrack、Cryptohaze GPU Rainbow Cracker 等，並提供大量定制的彩虹表下載。目前進展中的時間存儲折中攻擊專案還包括 Rainbowcrack 工程[29]和 A5/1 破解工程[25]等。

鑒於有如此之多的時間存儲折中攻擊方法實施方案，如何評價一個實施方案的好壞、如何在特定環境下選擇合適的實施方案、如何設定實施方案的適當參數，涉及到成功率、誤報、預計算時間、存儲空間、線上攻擊時間等多個要素的取捨，不是一個容易回答的問題。

2010 年，文獻[13]首次分析了 Hellman 方法和彩虹表方法中誤報對線上攻擊時間的影響；文獻[16]提出在固定相同成功率的情況下對實施方案公平的比較，考慮誤報對線上攻擊時間的影響，文獻[16]比較了平均情況下非完美表版本的 Hellman 方法、彩虹表方法、區分點表方法的攻擊效率，文獻[24]比較了完美表版本的 Hellman 方法、彩虹表方法、區分點表方法的攻擊效率。時間存儲折中攻擊方法的其它理論分析結果還包括文獻[15][21][22][23]等。

參考文獻

[1] G. Avoine, A. Bourgeois, and X. Carpent, "Discarding the endpoints makes the cryptanalytic time-memory trade-offs even faster," *IACR Cryptology ePrint Archive 2012* (2012), 683.

[2] G. Avoine, P. Junod and P. Oechslin, "Characterization and improvement of time-memory trade-off based on perfect tables," *ACM Transactions on Information and System Security (TISSEC)* 11, 4 (2008), 17.

[3] G. Avoine, P. Junod and P. Oechslin, "Time-memory tradeoffs: False alarm detection using checkpoints," In *Progress in Cryptology-INDOCRYPT 2005*, Springer, 2005, pp. 183-196.

[4] E. Barkan, E. Biham and A. Shamir, "Rigorous bounds on cryptanalytic time/memory tradeoffs," In *Advances in Cryptology-CRYPTO 2006*, Springer, 2006, pp. 1-21.

[5] A. Biryukov, "Some thoughts on time-memory-data tradeoffs," *IACR Cryptology ePrint Archive 2005* (2005), 207.

[6] A. Biryukov, S. Mukhopadhyay and P. Sarkar, "Improved time memory tradeoffs with multiple data," In *Selected Areas in Cryptography(2006)*, Springer, pp. 110-127.

[7] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers," In *Advances in Cryptology)-ASIACRYPT 2000*. Springer, 2000, pp. 1-13.

[8] A. Biryukov, A. Shamir and D. Wagner, "Real time cryptanalysis of a5/1 on a pc," In *Fast Software Encryption (2001)*, Springer, pp. 1-18.

[9] J. Borst, B. Preneel and J. Vandewalle, "On the time-memory tradeoff between exhaustive key search and table precomputation," In *Symposium on Information Theory in the Benelux (1998)*, Citeseer, pp. 111-118.

[10] J. Daemen and V. Rijmen, "The rijndael block cipher: Aes proposal," Computer Security Resource Center, National Institute of Standard Technology (1999).

[11] O. Dunkelman and N. Keller, "Treatment of the initial value in timememory-data tradeoff attacks on stream ciphers," *Information Processing Letters* 107, 5 (2008), 133-137.

[12] D. Eastlake and P.U. Jones, "secure hash algorithm 1 (sha1)," 2001.

[13] J. Hong, "The cost of false alarms in hellman and rainbow tradeoffs," *Designs, Codes and Cryptography* 57, 3 (2010), 293-327.

[14] J. Hong, K.C. Jeong, E.Y. Kwon, I.-S. Lee and D. Ma, "Variants of the distinguished point method for cryptanalytic time memory trade-offs," In *Information Security Practice and Experience*, Springer, 2008, pp. 131-145.

[15] J. Hong, G.W. Lee and D. Ma, "Analysis of the parallel distinguished point tradeoff," In *Progress in Cryptology-INDOCRYPT 2011*, Springer,2011, pp. 161-180.

[16] J. Hong and S. Moon, "A comparison of cryptanalytic tradeoff algorithms," *Journal of cryptology* 26, 4 (2013), 559-637.

[17] J. Hong and P. Sarkar, "New applications of time memory data tradeoffs," In *Advances in Cryptology-ASIACRYPT 2005*, Springer, *2005*, pp. 353-372.

[18] J. Hong and P. Sarkar, "Rediscovery of time memory tradeoffs," *IACR Cryptology ePrint Archive 2005 (2005)*, 90.

[19] J. Hong and P. Sarkar, "New applications of time memory data tradeoffs," In *Advances in Cryptology-ASIACRYPT 2005*, Springer, 2005,pp. 353-372.

[20] M. Hellman, "A cryptanalytic time-memory trade-off," *IEEE Transactions on Information Theory* 26, 4 (1980), 401-406.

[21] B.-I. Kim and J. Hong, "Analysis of the perfect table fuzzy rainbow tradeoff".

[22] J.W. Kim, J. Hong and K. Park, "Analysis of the rainbow tradeoff algorithm used in practice," *IACR Cryptology ePrint Archive 2013* (2013), 591.

[23] J.W. Kim, J. Seo, J. Hong, K. Park and S.-R. Kim, "High-speed parallel implementations of the rainbow method based on perfect tables in a heterogeneous system," *Software: Practice and Experience (2014)*.

[24] G.W. Lee and J. Hong, "A comparison of perfect table cryptanalytic tradeoff algorithms," *IACR Cryptology ePrint Archive 2012* (2012), 540.

[25] K. Nohl, "Attacking phone privacy," *BlackHat 2010 Lecture Notes (2010)*.

[26] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," In *Advances in Cryptology-CRYPTO 2003*, Springer, 2003, pp. 617-630.

[27] R. Rivest, "The md5 message-digest algorithm".

[28] D.E. Robling Denning, "Cryptography and data security," Addison-Wesley Longman Publishing Co., Inc., 1982.

[29] Rainbowcrack project, HTTP://project-rainbowcrack.com/ (2014/12/1).

[30] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater and J.-D. Legat, "A time-memory tradeoff using distinguished points: New analysis & fpga results," In *Cryptographic Hardware and Embedded Systems-CHES 2002*, Springer, 2003, pp. 593-609.