九州大学学術情報リポジトリ
Kyushu University Institutional Repository

ブルームフィルタを用いた高速RFID識別方式

野原, 康伸
九州大学大学院システム情報科学府

井上, 創造
九州大学附属図書館

安浦, 寛人
九州大学大学院システム情報科学研究院

https://hdl.handle.net/2324/8513

出版情報：コンピュータセキュリティシンポジウム2007 論文集 || 2007(10) || p301-306. 2007 (10), pp.301-306, 2007-11-02
バージョン：
権利関係：

ブルームフィルタを用いた高速 RFID 識別方式

野原　康伸 †　　井上　創造 ‡　　安浦　寛人 ∗

† 九州大学大学院 システム情報科学府　　‡ 九州大学附属図書館
∗ 九州大学大学院 システム情報科学研究院
819-0395 福岡市西区元岡 744 番地

{nohara,sozo,yasuura}@c.csce.kyushu-u.ac.jp

あらまし RFID は近年注目を浴びる自動認識技術であるが、第三者がユーザの行動を無断で追跡できてしまうというプライバシ問題が指摘されている。この問題に対し、大久保らによってハッシュチェイン方式が提案されたが、1 回の識別に要する時間が多い、もしくは大量のメモリが必要という課題があった。本稿では、事前計算した出力リストをブルームフィルタを用いて圧縮しておくことにより、少量のメモリ使用でハッシュチェイン方式の識別を高速化する手法を提案する。

A High-Speed Identification Scheme for RFID Using Bloom Filter

Yasunobu NOHARA†　　Sozo INOUE‡　　Hiroto YASUURA∗

†Graduate School of Information Science and Electrical Engineering, Kyushu University
‡Kyushu University Library
∗ Faculty of Information Science and Electrical Engineering, Kyushu University
744 Motooka Nishi-ku Fukuoka 819-0395 JAPAN

{nohara,sozo,yasuura}@c.csce.kyushu-u.ac.jp

Abstract As pervasive computing environments become popular, RFID tags are introduced into our daily life. However, there exists a privacy problem that an adversary can trace users' behavior by linking the tag's ID. Although a hash chain scheme can solve this privacy problem, the scheme needs a long identification time or a large amount of memory. In this paper, we propose an efficient identification scheme using Bloom filters. Our scheme provides a high-speed identification with a small amount of memory by storing pre-calculated results in Bloom filters.

1　はじめに

RFID(Radio Frequency IDentification) システムとは、無線通信が可能な小型 IC(RFID タグ) を用いて、人や物の識別をするシステムのことである。RFID タグを用いることにより現実世界の人や物に電子的な識別子をつけ、現実世界の情報とデータベース上の仮想世界とを対応づけることが可能となる。RFID システムは、どこにでも計算機が存在する環境を意味するユビキタスコンピューティングの根幹をなす基盤システムとして期待が大きい。

RFID システムでは無線により通信が行われるため、第三者がユーザに気づかれることなく無断で RFID タグ上の ID を収集することができる。このため、第三者が様々な場所にリーダを設置し、ID 等を紐付けすることにより第三者がユーザの行動を追跡できてしまうというプ

ライバシ問題がある。この問題の解決方法として、情報の送信元が同一の人物によるものであるかを判定できないというリンク不能性 (Unlinkability) を実現する方法がある。

大久保らによって提案されたハッシュチェーン方式 [1] はリンク不能性の一実現手法である。ハッシュ関数を用いた方式であるため、公開鍵暗号を用いた方式と比べてタグの計算負荷が軽く、コストの制約が非常に厳しい RFID タグに適した方式である。またハッシュチェーン方式は、タグがタンパリングされ秘密情報を盗まれたとしても、過去に遡ってリンク不能性が破られることは無いというフォワードセキュアと呼ばれる性質を持つ。しかしながら、この方式は 1 回の識別毎に大量のハッシュ計算を必要とする、もしくは事前に計算したハッシュ値を保存する為に大量のメモリが必要という問題があった。

本稿では、事前計算した出力リストをブルームフィルタを用いて圧縮しておくことにより、少量のメモリ使用でハッシュチェーン方式 [1] の識別を高速化する手法を提案する。

本稿の構成は以下の通りである。2 章でハッシュチェーン方式について、3 章でブルームフィルタについて説明を行う。4 章でブルームフィルタを用いた識別の高速化手法を提案する。5 章で関連研究を紹介し、提案手法との比較を行う。6 章で本稿をまとめる。

2 ハッシュチェーン方式

ハッシュチェーン方式 [1] は、大久保らによって提案されたハッシュ関数を用いた RFID プライバシ保護の一手法である。

N 個の RFID タグがあるシステムを考える。各 RFID タグは、タグに固有の ID_i と乱数の種を保持している。乱数の種の初期値を $r_{i,1}$ とする。サーバは、全タグの ID_i と $r_{i,1}$ のデータを保持している。

RFID タグ i が j 回目に出力するタグ出力 $o_{i,j}$ は、$o_{i,j} = H(ID_i \| r_{i,j})$ で与えられる。このとき乱数の種は、$r_{i,j+1} = G(r_{i,j})$ に更新される。ただし、H, G は異なる 2 つの一方向性ハッシュ関数である。

リーダが 1 つしか存在しないシステムの場合、タグ出力から識別を行う際にサーバは、全ての $1 \leq i \leq N$ について $o_{i,l_i} = H(ID_i \| G^{(l_i-1)}(r_{i,1}))$ を計算し、タグ出力とマッチする i を探し出すことにより識別を行える。ここで l_i は、リーダがタグ i の出力を最後に観測したときにおけるタグ i の出力回数である。

しかしながら、攻撃者のものを含めた複数のリーダが存在するシステムの場合、他のリーダによってタグの出力が読まれると、サーバとタグの間の乱数の種の同期がずれてしまうため、識別できなくなってしまう。

そこで、M という同期ずれ許容値を定める。タグ出力から識別を行う際にサーバは、全ての $1 \leq i \leq N, l_i + 1 \leq j \leq l_i + M$ について $o_{i,j} = H(ID_i \| G^{(j-1)}(r_{i,1}))$ を計算し、タグ出力とマッチする i, j の組を探し出す。i が求める答えである。この方式を**逐次計算方式**とよぶこととする。

またあらかじめ各 i について出力リスト

$$O_i = \{o_{i,l_i+1}, \cdots, o_{i,l_i+M}\}$$

を計算し、メモリに保存しておくことにより、逐次計算方式で必要だった 1 回の識別ごとの $o_{i,j}$ の計算を省くことができる。この方式を **LUT 方式**とよぶこととする。

タグ出力 $o_{i,j} = H(ID_i \| r_{i,j})$ は毎回変化する $r_{i,j}$ を含んでいるため、毎回可変である。また一方向性関数の性質より、攻撃者が $o_{i,j}$ から ID_i を取得することは困難である。したがって、ハッシュチェーン方式はリンク不能性を満たす。

また、$r_{i,j}$ がタンパリング等により盗まれたとしても、一方向性関数の性質上、$r_{i,j'} (j' < j)$ を取得することは困難である。したがって、過去に遡ってリンク不能性が破られることは無いというフォワードセキュア性を満たす。

しかしながら、ハッシュチェーン方式は、逐次計算方式の場合は 1 回の識別に要する平均ハッシュ計算量が $MN/2$ 回、LUT 方式の場合はメモリが $MN|s|$ 必要であり、NM が大きなシステムに適用することが難しいという問題点があった。ただし、$|s|$ は一方向性ハッシュ関数の出力長を表す。

なお、$o \in O_i$ が与えられたときに正しい i を出力することを識別成功、識別が成功する確率を識別成功率と呼ぶこととする。逐次計算方式、LUT 方式共に識別成功率は 100%である。

3 ブルームフィルタ

ブルームフィルタとは、Burton H. Bloom が考案した空間効率の良い確率的データ構造であり、任意のデータ d が集合のメンバー X に含まれるかどうかの判定に用いられる [2]。判定の結果、$d \in X$ と判断された場合は Positive、$d \notin X$ と判断された場合は Negative という結果が出力される。

ブルームフィルタでは、データ $d' \notin X$ を Positive と誤判定する False Positive が発生することはあるが、データ $d \in X$ に対して Negative と誤判定する False Negative は発生することはない。したがって、ブルームフィルタは $d \in X$ であるかを判定する時に、検索の対象を絞り込むスクリーニングの手段として使うことができる。

オリジナルのブルームフィルタ [2] には X から要素を削除する機能は無いが、Fan らによって提案されたカウンティング・ブルームフィルタ (CBF) は、削除が可能となっている [3]。以下、ブルームフィルタといえば CBF のことを指すものとする。

3.1 機能

長さ m の配列 $BLOOM$ を用意する。$BLOOM$ の各要素にはカウンタ値 (0 以上 15 以下) が格納される[1]。また、出力域が $[1, m]$ となるような独立した k 個のハッシュ関数 $h_i : \{0,1\}^* \rightarrow [1, m] (1 \leq i \leq k)$ を用意する。なお、h_i は一方向性を満たす必要は無い。

以下に、ブルームフィルタで用いられる関数とその機能の一覧を示す。なおスペースの都合上、各関数の具体的なアルゴリズムについては省略する。

- $InitBloom(BLOOM)$: フィルタを初期化する。$X = \phi$ となる。
- $AddBloom(x, BLOOM)$: 集合 X に要素 x 追加する。
- $SearchBloom(d, BLOOM)$: データ d が $d \in X$ であるかを判定する。戻り値は、Positive または Negative である。
- $DeleteBloom(x, BLOOM)$: 要素 $x \in X$ を集合 X から削除する。

3.2 n, m, k とエラー率の関係

n を集合の要素数とすると、エラー率 ϵ(False Positive 発生率) は、

$$\epsilon = [1 - (1 - 1/m)^{kn}]^k \approx (1 - e^{-kn/m})^k$$

で与えられる。m, n が与えられているとき、

$$\hat{k} = \frac{m}{n} \ln 2$$

とすればエラー率は最小となり、その時のエラー率は、

$$\hat{\epsilon} = 0.5^{\hat{k}} \approx 0.6185^{m/n}$$

となる。

4 ブルームフィルタを用いた識別

本章では、ブルームフィルタを用いた高速な識別手法について提案する。提案手法では、事前計算した出力リストをブルームフィルタを用いて圧縮しておくことにより、少量のメモリ使用で高速な識別を実現する。

4.1 事前計算

識別のための事前計算として、出力リストを圧縮する方法について述べる。

最初にブルームフィルタを N 個用意し、各タグと各ブルームフィルタを 1 対 1 に対応させる。以下、タグ i に対応するブルームフィルタ配列を $BLOOM_i$ と表す。$BLOOM_i$ には、集合 O_i の各要素が格納される。

[1] カウンタ値が 16 以上の値をとる可能性もあるが、確率的に十分無視できる [3]

事前計算の手順を擬似コードで記述したものを Algorithm 1 に示す。ただし、$make_output(i,j)$ は、i 番目のタグの j 回目の出力を求める関数である。

Algorithm 1 事前計算

Input: $BLOOM_1, \cdots, BLOOM_N, l_1, \cdots, l_N$
Output: $BLOOM_1, \cdots, BLOOM_N, l_1, \cdots, l_N$

```
for i = 1 to N do
    l_i ← 0
    InitBloom(BLOOM_i)
    for j = 1 to M do
        id ← make_output(i, j)
        AddBloom(id, BLOOM_i)
    end for
end for
```

出力リストの圧縮に必要なハッシュ計算は、N 個ある各 RFID タグについてそれぞれ M 個の ID を生成するので、NM 回必要である。また必要なメモリ量は、N 個ある各ブルームフィルタにそれぞれ M 個の ID を格納するので、$4mN = 4NM \log_{0.6185} \epsilon$[bit] である。

4.2 識別

前節で作成した出力リストが格納されたブルームフィルタを用いて、識別を行う方法について述べる。

タグ出力 d から ID を検索するため、N 個ある各ブルームフィルタに対して、d が要素として含まれているかどうかの判定を行う。Positive を出力するブルームフィルタが見つかれば、対応する i を得ることで、識別を行うことができる。

ただし、ブルームフィルタには False Positive の可能性があるため、複数のブルームフィルタが Positive を出力する場合がある。この場合は、Positive を出力したブルームフィルタに対応する各 ID_i について、総当りの検索を行うことにより i を求めることができる。このとき同時に何回目の出力であるかを表す j についても求めることができる。

識別アルゴリズムを擬似コードで記述したものを Algorithm 2 に示す。

Algorithm 2 識別

Input: $d, BLOOM_1, \cdots, BLOOM_N, l_1, \cdots, l_M$
Output: (i,j) or Negative

```
A ← φ
for i = 1 to N do
    if SearchBloom(d, BLOOM_i) = Positive then
        A ← A ∪ {i}
    end if
end for
for all i ∈ A do
    for j = l_i + 1 to l_i + M do
        if d = make_output(i, j) then
            return (i, j)
        end if
    end for
end for
return Negative
```

Positive を出力するブルームフィルタには、正しくマッチする 1 つのフィルタと False Positive によりマッチしてしまうフィルタの 2 種類があり、その平均個数は $1 + \epsilon(N-1)$ である。1 つの ID についての総当り検索に必要なハッシュ計算の平均値は $M/2$ であるので、平均ハッシュ計算回数は、$[\epsilon(N-1)+1]M/2$ 回となる。

ブルームフィルタには False Positive が起こる可能性が無いので、識別成功率は 100%である。

4.3 出力リストの更新

前節の識別によって得られた (i,j) を用いて、l_i の値の更新と出力リスト O_i が格納されている $BLOOM_i$ の更新を行う。

識別前の $O_i^{old} = \{o_{i,l_i^{old}+1}, \cdots, o_{i,l_i^{old}+M}\}$ と更新後の $O_i^{new} = \{o_{i,j+1}, \cdots, o_{i,j+M}\}$ の間には

$$O_i^{new} = O_i^{old} - \{o_{i,l_i^{old}+1}, \cdots, o_{i,j}\} \cup \{o_{i,l_i+M+1}, \cdots, o_{i,j+M}\}$$

という関係が成り立つ。したがって、出力リストの更新では $BLOOM_i$ より $\{o_{i,l_i^{old}+1}, \cdots, o_{i,j}\}$ の削除と $\{o_{i,l_i+M+1}, \cdots, o_{i,j+M}\}$ の追加を行う必要がある。また l_i を、$l_i^{new} = j$ と更新する。

上記アルゴリズムを擬似コードで記述したものを Algorithm 3 に示す。

Algorithm 3 リストの更新

Input: $BLOOM_i, l_i, i, j$

Output: $BLOOM_i, l_i$

1: **for** $k = l_i + 1$ to j **do**
2: $id \leftarrow make_output(i, k)$
3: $DeleteBloom(id, BLOOM_i)$
4: $id \leftarrow make_output(i, k + M)$
5: $AddBloom(id, BLOOM_i)$
6: **end for**
7: $l_i \leftarrow j$

$l_i^{new} - l_i^{old}$ の平均値は $M/2$ であるので、追加する ID と削除する ID を生成するのには、それぞれ $M/2$ 回の平均ハッシュ計算が必要である。しかしながら、削除する ID については識別時に生成を行っている。したがって、識別時に生成した ID を再利用すれば、必要な総平均ハッシュ計算は $M/2$ となる。

5 比較

本章では、関連研究を紹介し、提案手法との比較を行う。

5.1 関連研究

Avoine らによって Time-memory Tradeoff を用いたハッシュチェーンの高速化手法が提案されている [4]。Avoine 方式は、暗号解読などで使われる Time-memory Tradeoff 方式を用いることにより、識別に必要な計算量とメモリ量の調整ができるようになっている。このため提案方式と同じように、システムの環境に応じた柔軟な運用が可能である。

しかしながら、Avoine 方式は 1) 事前計算量が $O(NM^2)$ 必要である、2) 識別成功率は 100% よりも低いという点に加えて、3) 頻繁な事前計算が必要という問題点が存在する。これは Avoine 方式では、提案手法と異なり出力リストを随時更新することができないため、頻繁に出力リストの全更新を行なう必要があるからである。

5.2 比較

表 1 に提案手法と LUT 方式・逐次計算方式、Avoine 方式の平均ハッシュ計算回数やメモリー量等を比較したものを示す。ここで、c はメモリパラメータである。また [] 内の値は、ブルームフィルタで用いるハッシュ計算回数である。ブルームフィルタのハッシュ関数には、一方向性が要求されないのでハッシュチェーンで用いる一方向性ハッシュ関数よりも計算負荷は軽いと考えられる。

5.3 実験

$N = 102400, M = 256$ とした場合における、提案方式・LUT 方式・逐次計算方式の実行時間を計測した。提案方式は、ϵ を変えた 4 パターンについて計測を行った。

実験に使用する PC のスペックは、CPU: Intel Core Duo T2300(1.66GHz, 2MB L2 Cache), メモリ: 1GB である。一方向性ハッシュ関数には SHA1($|s| = 160$[bit]) を用いた。

表 2 に各方式の処理時間の計測結果を示す。

提案方式と LUT 方式を比べると、提案方式の方が事前計算時間が多いが、より少ないメモリで高速な識別ができている事が分かる。また 1 回の識別を行う毎にサーバで必要である「識別+出力リストの更新」の処理も高速である。

次に提案手法と逐次計算方式を比べると、提案方式のほうがメモリ・事前計算の点で不利ではあるが、大幅な識別の高速化を実現していることが分かる。

最後に提案方式同士を比べると、メモリ使用量が一番大きい 400MB のときではなく、300MB のときに識別及び出力リストの更新処理が最速であることが分かる。これはメモリ使用量を増やすと、ブルームフィルタのエラー率が減り、一方向性ハッシュ計算が減る反面、ブルームフィルタにおけるハッシュ計算が増える為、両者のバランスの取れた 300MB の時に処理時間が最短になったものと考えられる。

表 1: 各方式のメモリ・計算回数比較

方式名	メモリ [bit]	平均ハッシュ計算回数 [回]			識別成功率 [%]
		識別	更新	事前計算	
LUT	$NM\|s\|$	0	$M/2$	NM	100
提案	$4NM\log_{0.6185}\epsilon$	$\{\epsilon(N-1)+1\}M/2$ $[+\log_{0.5}\epsilon]$	$M/2$ $[+M\log_{0.5}\epsilon]$	NM $[+NM\log_{0.5}\epsilon]$	100
Avoine	$cN\|s\|$	$\frac{3^3M^3}{2^2c^3}$	不可	$NM^2/2$	99.9
逐次	0	$NM/2$	0	0	100

表 2: 処理時間の計測結果

方式名	メモリ [MB]	識別 [ミリ秒]	更新 [ミリ秒]	識別+更新 [ミリ秒]	事前計算 [秒]
LUT	500	307.5	0.139	307.6	25.3
提案	400	12.8	0.865	13.7	84.7
	300	11.8	0.725	12.6	71.0
	200	14.7	0.551	15.2	54.9
	100	162.7	0.414	163.1	41.1
逐次	0	12958.0	0.000	12958.0	0.0

6 おわりに

本稿では、事前計算した出力リストをブルームフィルタを用いて圧縮しておくことにより、ハッシュチェーン方式の識別を高速化する手法を提案した。提案手法は、識別に必要な計算量とメモリ量の調整を行うことができ、システムの環境に応じた柔軟な運用が可能である。また従来の手法ではできなかった出力リストの随時更新が可能であり、事前計算の頻度を抑えることが可能となった。

謝辞

本研究は文部科学省科学研究費補助金若手研究 (A)「信頼性を実現する RFID 情報システムの研究」(課題番号 18680009) による。

参考文献

[1] Ohkubo, M., Suzuki, K. and Kinoshita, S.: Hash-Chain Based Forward-Secure Privacy Protection Scheme for Low-Cost RFID, in *2004 Symposium on Cryptography and Information Security – SCIS2004*, Vol. 1, pp. 719–724 (2004).

[2] Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM*, Vol. 13, No. 7, pp. 422–426 (1970).

[3] Fan, L., Cao, P., Almeida, J. and Broder, A. Z.: Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol, *IEEE/ACM TRANSACTIONS ON NETWORKING*, Vol. 8, No. 3, pp. 281–293 (2000).

[4] Avoine, G. and Oechslin, P.: A Scalable and Provably Secure Hash-Based RFID Protocol, in *2nd International Workshop on Pervasive Computing and Communications Security – PerSec2005*, pp. 110–114, IEEE Computer Society Press (2005).