なにか

須賀 祐治[1]

概要：ワラ納豆型ワンタイムパスワード認証方式という新しいコンセプトを提案し，その適用について報告する．本方式は Lamport-like なワンタイムパスワード方式の拡張であるとともに，認証を行うと同時にクライアント（Prover）からサーバ（Verifier）に対して任意のデータを送付することが可能となる．Markle Tree と L 分木をジョイントしたハッシュ連鎖構造を用いており，これがワラ納豆のような形を持つことから，ワラ納豆型ワンタイムパスワード認証方式と名付けられた．このとき，ハッシュ関数には一方向性のみを仮定しており，落とし戸は不要であることから SHA-2/3 などの dedicated なハッシュ関数を用いることができるため，高速な実装が可能である．

キーワード：ワンタイムパスワード, Lamport スキーム, マークル木, ナット

NAtto-style New Incredible Kind of one-time Authentication schemes (NANIKA)

Yuji Suga[1]

Abstract: This paper proposes Natto-style new kind of one-time password schemes and describes their applications. A proposal is an extension of the Lamport-like scheme and has an advantage that we could send data from the client (prover) to the server (verifier) while authentication phase. The proposal uses jointed hash chain with Markle tree and binary tree, this kind of jointed tree seems the straw wrapper, so we call the proposals Natto-style one-time password schemes. We assume that the hash function has only one-way feature, so we can use dedicated cryptographic hash function such as SHA-2/3, this means that we do not need the trapdoors in the hash function, so we realize efficient implementations.

Keywords: One-time Password, Lamport's scheme, Markle Tree, Natto

1. やりたいこと

Lamport-like なワンタイムパスワード方式拡張を試みる．提案方式では認証を行うと同時にクライアント（Prover）からサーバ（Verifier）に対して任意のデータを送付することが可能となる．このデータは事前に準備したデータではなく，認証を行う際にユーザがその都度自由に選択することができる．

2. コンセプト

図 1 は本論文で提案する認証方式のコンセプトを示したものである．全体としてワラ納豆の形をしていることからワラ納豆型ワンタイムパスワード認証（納豆認証）と名付けられた．図 1 のグラフは右半分は L 分木であり，そのリーフを基に Merkle 木を構成し，そのルートを終点（Destination）と呼ぶ．各ノードはある区間の値を持ち，有向のエッジで結ばれている．エッジを通すごとに暗号学的ハッシュ関数（一方向性を有する関数）を用いてノードに割り振られた値は変化すると考えてよい．また一方向性を有することからエッジの先のノードから基のノードには戻せないことに注意する．このように起点から終点のグラフ構造をナットと呼ぶことにする．またナットに含まれる各ノードのことをナット粒と呼ぶこともある．

[1] 株式会社インターネットイニシアティブ
Internet Initiative Japan Inc.

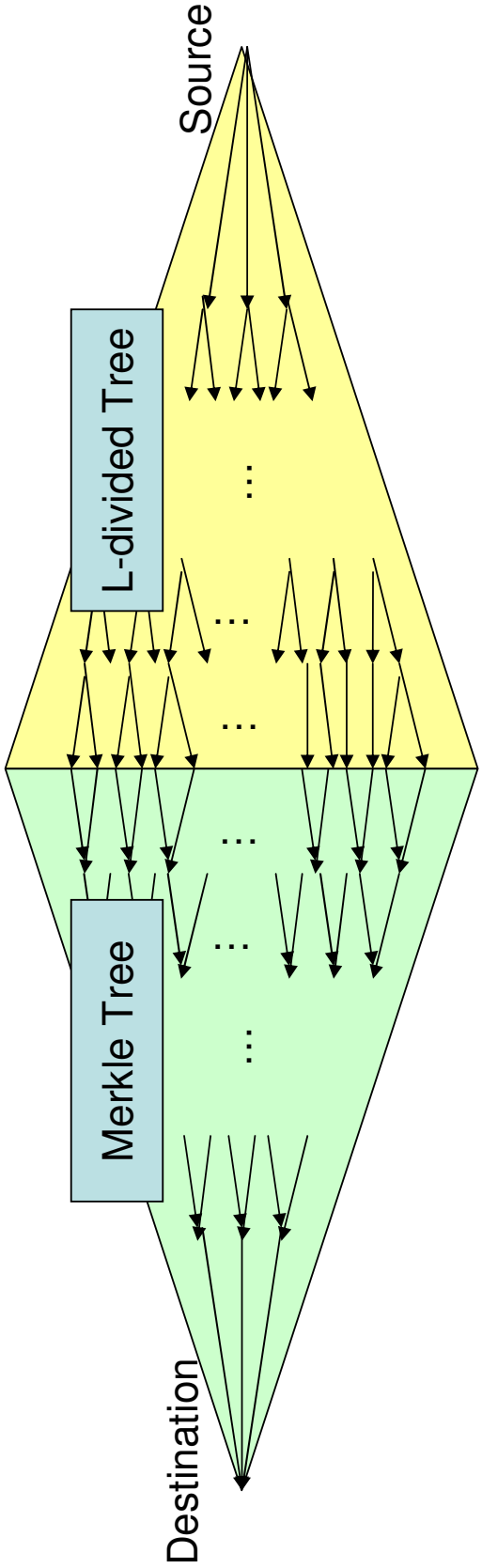

図 1 ワラ納豆型ワンタイムパスワード認証方式のコンセプト

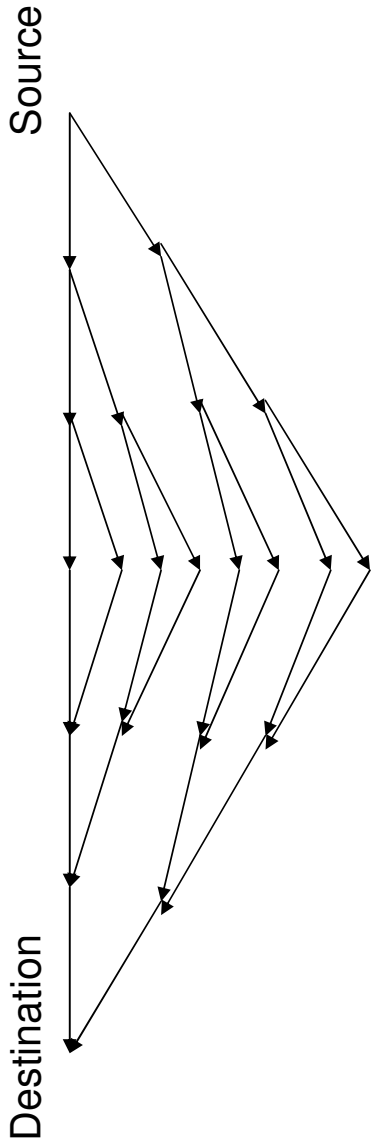

図 2 ワラ納豆型ワンタイムパスワード認証方式の例 ($L = 2, d = 3$)

本方式は Lamport-like なワンタイムパスワード方式の拡張である．Lamport スキームはハッシュ連鎖を一つ一つ解きほぐすことで得られたワンワイムパスワードをクライアントが送信し，サーバでの確認によりユーザ認証を行う方式である．本提案方式でも同様のスキームを用いることで認証を行う．

本方式ではユーザ認証を行うと同時にクライアント（Prover）からサーバ（Verifier）に対して任意のデータを送付することが可能となる．

ハッシュ関数には一方向性のみを仮定しており，落とし戸は不要であることから SHA-2/3 などの dedicated なハッシュ関数を用いることができるため，高速な実装が可能である．

図 2 はナットでも単純な表現である，次元 $L = 2$，深さ

$d = 3$ の 1 例を示したものである．始点から深さ 3 のバイナリツリーを構成することによって L 分木と Merkle 木の境界部分のノードは $8(= 2^3)$ 個存在することが分かる．Merkle 木を構成することで収束され，最終的には 1 点のノードである終点に帰着していく．

3. ナットの構成 ＝ワラ納豆の右を縛って粒を入れていく

前章を踏まえ，よりフォーマルな形でナットを定義していく．始点と終点を含むナット粒（ナットの各ノード）は区間 $[0, 2^m - 1]$ の元を持つ．ここで m はセキュリティパラメータであり，ナット粒が持つ元を算出するために用いられるハッシュ関数の出力長が概ね用いられることを想定する．例えば SHA-256 では $m = 256$ である．

3.1 拡散部

ナットは 2 つの部分で構成されており，始点から終点に向けて各ナット粒の元が計算可能である．まず起点を基にして L 分木のノードに拡散する「拡散部」では，親ノードから以下のような計算を行うことでノード粒の元を算出する．親ノードの元 p から L 個のノードの元 $p_i (i = 1, \ldots, L)$ を算出：

$$p_i := H(p||q_i)$$

このとき q_i はクライアントとサーバで共有されているものとする．公開パラメータとして考える場合と非公開パラメータとする場合でアプリケーションが変わってくる．q_i を公開パラメータとする場合には，各ナット粒の元を算出する際に異なる L 個のハッシュ関数と準備していると考えてよい．一方で q_i を非公開にするケースでは，第 3 者が認証時に発生するデータを閲覧しても関係性が分からない，つまり第 3 者が検証できない．これは認証のためだけに用いるケースでは安全性が高い利用方法と考えられる．一方で，後述するクライアントからサーバに対して認証と同時にデータを送信するシチュエーションでは，このデータを秘匿したいのか，敢えて公開して後日第 3 者が検証できるようにするのかで変わってくる．特に後者のケースでは，ビットコインなどのように第 3 者がトランザクションを閲覧できるようにする等，適用するアプリケーションの要件に依存する．

3.2 収束部

拡散部で生成されたナット粒から Merkle 木を構成する「収束部」では以下のような計算を行う．L 個の子ノードの元 $p_i (i = 1, \ldots, L)$ から親ノードの元 p を算出：

$$p := H(p_1|| \ldots ||p_L)$$

3.3 例：次元 $L = 2$，深さ $d = 3$

図 2 で示した構造において，拡散部と収束部の境界ノードは 8 点あり，これらを $r_1, \ldots, r_8$ とすると，起点の元 s から以下のように算出される．

- $r_1 := H(H(H(s||q_1)||q_1)||q_1)$
- $r_2 := H(H(H(s||q_1)||q_1)||q_2)$
- $r_3 := H(H(H(s||q_1)||q_2)||q_1)$
- $r_4 := H(H(H(s||q_1)||q_2)||q_2)$
- $r_5 := H(H(H(s||q_2)||q_1)||q_1)$
- $r_6 := H(H(H(s||q_2)||q_1)||q_2)$
- $r_7 := H(H(H(s||q_2)||q_2)||q_1)$
- $r_8 := H(H(H(s||q_2)||q_2)||q_2)$

さらに，ここから Merkle 木を構成すると最終的に終点の元 t は以下のようになる．

$$t := H(H(H(r1||r2)||H(r3||r4))||H(H(r5||r6)||H(r7||r8)))$$

3.4 ナットの連鎖

一旦，1 つのナットの構成が終わったら，得られた終点を始点としてさらにナットを構成することが可能である．複数のナットを連鎖させる際に，一つ一つのナットの次元 L や深さ d はそれぞれ異なるパラメータを用いてもよい．この場合，それぞれのナットの次元・深さをナットが変更になる度に共有しておく必要が出る．

こうして複数のナットを構成し，最終ナットの終点をサーバと共有することで認証のセットアップを終えることとなる．

4. ワンタイム認証フェーズ ＝ワラを解いて左から食べていく

クライアントは複数ナットの構成が終わったら，サーバには最終ナットの終点を共有している状況にある．このとき，Lamport 認証方式と同様に一つずつハッシュチェーンを解いていく，つまり，一つ前の状態に戻すことになる．本提案方式では，一つ解いていくと L の選択肢がある．ここに，どの状態に戻すかによって，クライアントからサーバにデータを送信することとなる．前半（左側）である収束部での解き方と，後半（右側）である拡散部での解き方が異なるため，それぞれ別節にて詳細に説明していく．

4.1 一度に送信できる情報量

いずれの場合も Lamport 方式とは異なり，一気に複数のフェーズに戻すことで一度の認証に多くのデータを共有することも可能である点に留意する．Lamport 方式では毎回の認証そのものには違いがなく平坦なものであったが，認証時にクライアントがドラスティックに振る舞うことで様々なアプリケーションが生み出される可能性がある．例えば，ユーザ認証と同時にプリペイドされた口座・ポイン

ト財布から課金することが考えられる．L は通常 2^l の形式であることを想定すると，1 つ状態を戻す際に $log(L) = l$ ビットの情報を送信できることから c 分状態遷移することで $c \cdot l$ ビットの情報を送信することができる．

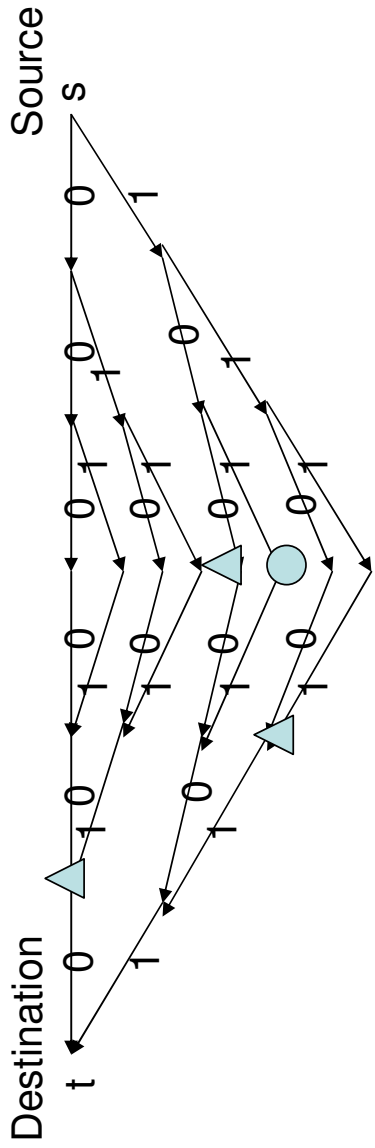

図 3 図 2 におけるユーザ認証の利用例

この例においては，サーバには三角で表現したノードの元を送ることで検証することが可能となる．Merkle 木の典型的な利用例であるため詳細な説明は省略する．ただし，このとき検証用データと遷移したいノードデータが混同すると送信データが正確に送れないことから，データの順番に関する取り決めは細心の注意が必要である．

4.3 拡散部

収束部との境界点から当該ノードに遷移するためにパスに位置づけされる各ナット粒の元を開示することが，単純にハッシュチェーンを辿っていることと同様である．また境界点ではない途中のノードからも同じようにパス上のナット粒の元を開示していくことでデータ送信が可能である．

5. 安全性についての簡単な考察

ハッシュチェーンを辿る点を鑑みると，拡散部の処理については Lamport 方式の安全性に帰着することができる．また収束部についてもハッシュ関数の一方向性が安全性に直結することから，現時点では初期的考察では問題ないと考えられる．ただしプロトコルやデータフォーマット等に malleability が存在するケースでは軽微な脆弱性が考えられるが，これまでの知見を用いることで多くのケースでは問題ないと考えられる．

サーバ・クライアントで共有する

6. 実装に関する留意点

ハッシュ関数には一方向性のみを仮定しており，落とし戸は不要であることから SHA-2/3 などの dedicated なハッシュ関数を用いることができるため，高速な実装が可能である．ただし落とし戸付きにすることで，さらに面白いアイデアが生まれる可能性もあることから，現時点では SHA-Family による実装を想定しているに過ぎず，場合によっては公開鍵暗号ベースのハッシュ関数が一時的に，つまりすべてのハッシュ関数計算には用いなくとも，あるタイミングで利用することも考えられる．

7. まとめ

ワラ納豆型ワンタイムパスワード認証方式という新しいコンセプトを提案し，その適用について報告した．この類のワンタイムパスワード方式にニーズがあるかについての背景調査と，新しいアプリケーションの創出を検討したい．

参考文献

[1] Leslie Lamport, "Password Authentication with Insecure Communication", Communications of the ACM 24.11, 770-772, 1981.

[2] Ralph Merkle, "Secrecy, Authentication and Public Key Systems/ A certified digital signature", Ph.D. dissertation, Dept. of Electrical Engineering, Stanford University, 1979.

[3] Michael Szydlo, "Merkle Tree Traversal in Log Space and Time", EUROCRYPT2004.

[4] http://www.hiyama710.com/annai.html (accessed at 12 August, 2016)