Autenticação e o Problema do Logaritmo Discreto

Fábio Borges[1], **Pedro Carlos da Silva Lara**[1]

[1]Coordenação de Sistemas e Redes – Laboratório Nacional de Computação Científica
Av. Getulio Vargas, 333, Quitandinha CEP: 25651-075 Petrópolis - Rio de Janeiro.

{borges,pcslara}@lncc.br

Abstract. *In this paper we propose an authentication scheme based on the difficulty to solve the discrete logarithm problem. This proposal differs from the current methods of authentication based in it hash functions. However, it does not substitute such functions, as they are used in other applications, for example, in the digital signature process. Our method has two great advantages, there is no collision and it isn't necessary to store the user's login.*

Resumo. *Neste artigo propomos um esquema de autenticação baseado na dificuldade de resolver o problema do logaritmo discreto. Esta proposta diferencia dos atuais métodos de autentica baseados em função de hash. No entanto, não substitui as tais funções, pois elas são usadas em outras aplicações, por exemplo, no processo de assinatura digital. Nosso método tem duas grandes vantagens, não há colisão e não é necessário armazenar o login do usuário.*

Palavras-chave: Autenticação, Criptografia, Segurança, Protocolos.

1. Introdução

Muitos meios de autenticações usados atualmente são baseados em funções de *hash*, tais funções permitem que uma *string* de tamanho indefinido seja levada a outra *string* de tamanho fixo. Elas podem, por exemplo, levar um arquivo de qualquer tamanho em uma seqüência de *bits* de tamanho fixo, ou ainda levar uma senha de acesso a um sistema em uma outra seqüência de *bits* com o mesmo tamanho. Atualmente estas funções estão sofrendo muitos ataques criptográficos, por isto fomos motivados a propor uma solução baseada no problema do logaritmo discreto. Fazemos uma revisão das principais funções de *hash* e do problema do logaritmo discreto, então propormos dois métodos de autenticação baseados no problema do logaritmo discreto. Após a apresentação dos métodos fazemos uma análise da segurança.

2. Funções de Hash

Atualmente as duas principais funções de *hash* são SHA e MD5. O SHA (*Secure Hash Algorithm*) [NIST 2002] foi criado pelo NSA (*National Security Agency*) e desenvolvido pelo NIST (*National Institute of Standards and Technology*), foi publicada com um FIPS (*Federal Information Processing Standard*) em 1993. Mais precisamente o SHA foi conhecido como o FIPS PUB 180. Em 1995, esta publicação foi revisada e ganhou a identificação de FIPS PUB 180–1 a qual é conhecida genericamente como SHA–1. Ele produz um valor de *hash* de 160 *bits* a partir de um tamanho arbitrário da mensagem, mas já existe SHA de até 512 *bits*. O MD5 foi desenvolvido em 1991 por Ronald Rivest para suceder ao MD4 que tinha alguns problemas de segurança. O MD5

(*Message–Digest algorithm 5*) [Rivest 1992] é um algoritmo de *hash* de 128 *bits* criado pela RSA Data Security, Inc., algumas distribuições Linux usam o MD5 no módulo PAM [Fernandes and Reddy 2002]. O método de verificação é feito pela comparação das duas *strings* de *hash*, uma da base de dados e a outra da senha digitada na tentativa de conexão. O MD5 também é usado para verificar a integridade de um arquivo através, por exemplo, do programa *md5sum*, que cria um *hash* de um arquivo.

Dado o resultado de uma função de *hash* é impossível obter a *string* inicial. Logo uma função de *hash* não admite inversa, pois não é injetora, se existisse tal inversa, a função de *hash* seria uma excelente compactadora de dados, porém desde o início da Teoria da Informação ficou claro que a taxa de compactação não poderia ser tão alta [Shannon 1948]. No entanto a propriedade de não ter inversa é o que mais causa preocupação nestas funções, o fato de ser matematicamente provado a possibilidade que vários elementos da imagem possam levar no mesmo elemento do domínio, chamamos isto de colisões. Assim uma função de *hash* destrói a mensagem de entrada, ou seja, para uma única mensagem autenticada por uma função de *hash* existem várias (infinitas) mensagens de entrada que levam a esta mensagem autenticada. É importante observar que as funções de *hash* não possuem chaves, fazendo uma ingênua analogia, cada pessoa possui uma impressão digital que a distingue das outras pessoas, no caso de uma função de *hash*, esta identifica probabilisticamente uma mensagem de tamanho arbitrário das outras mensagens. Podemos definir formalmente o conceito de colisão: considere h uma função de *hash* qualquer e x, x' duas *strings* de entrada de tamanho indefinido, tal que $x \neq x'$, uma colisão é caracterizada quando temos $h(x) = h(x')$.

É importante ressaltar que as funções de *hash* são projetadas de forma que seja praticamente impossível criar uma mensagem que resulte em um *hash* definido anteriormente, ou criar duas mensagens diferentes que resultem um hash de um mesmo valor. Esta é a grande diferencial de uma boa função de hash. Uma outra possibilidade é tentar um ataque de força bruta contra a função de hash, isso quer dizer, poderíamos gerar *hash* de diversas mensagens, uma após a outra, procurando por uma mensagem que resultasse em um valor de *hash* particular, ou duas mensagens que resultem em um mesmo valor de *hash*. Observe que a taxa de sucesso deste tipo de ataque depende diretamente do tamanho da saída da função de *hash*.

A probabilidade de se encontrar uma colisão aleatoriamente é quase nula, no entanto, estão sendo desenvolvidos métodos para encontrar colisão no MD5 [Liang and Lai 2005, Stevens 2006, Dobbertin 1996], no SHA1 [Wang et al. 2005], ou métodos mais gerais [Wang and Yu 2005].

As colisões estão motivando a criação de novas funções de *hash*, sendo aberto um concurso através do NIST para o novo padrão de *hash* dos EUA. Tal crise nestas funções nos motivou a propor um modelo diferente de autenticação.

Uma colisão em um sistema de autenticação faz com que um usuário possa ter acesso com o *login* de outro, mesmo sem saber a senha. Pois a senha verdadeira s e a falsa s' têm o mesmo *hash* $h(s) = h(s')$. Nossa proposta, baseada no Problema do Logaritmo Discreto (PLD), garante que não existe uma senha falsa que tenha o acesso autorizado.

3. Problema do Logaritmo Discreto

A primeira prática de criptografia assimétrica foi proposta por Whitfield Diffie e Martin Hellman [Diffie and Hellman 1976], este algoritmo se baseia diretamente na dificuldade de se resolver o PLD e foi arquitetado para que dois usuários estabelecessem uma chave simétrica através de um canal de comunicação inseguro. Desde então muitos criptossistemas assimétricos usam esta técnica como base de segurança de seus algoritmos, tendo em vista a dificuldade de resolver o PLD [Odlyzko 2000]. Taher ElGamal [ElGamal 1985] incorporou este problema e descreveu um protocolo de chave assimétrica para cifrar e fazer assinaturas digitais, em 1985, Victor Miller [Miller 1986] e Neal Koblitz [Koblitz 1987] propuseram, independentemente, o uso de curvas elípticas em criptografia, cuja base deste criptossistema está exatamente na dificuldade de resolver o PLD na estrutura de grupo formada por uma curvas elípticas.

Considere a equação $\beta = \alpha^a \mod p$, de modo geral, mesmo conhecendo α, β e p encontrar a é um problema computacionalmente inviável, neste trabalho incorporamos a dificuldade do PLD em autenticação de sistemas.

Se estivermos trabalhando com $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ as variáveis devem ser na ordem de 1024 *bits*, no entanto se trabalharmos com uma curva elíptica basta 160 bits [Koblitz and Menezes 2004]. A criptografia com curvas elípticas tem demonstrado ser um ótimo método, visto que necessita de uma chave consideravelmente menor que as usadas em alguns clássicos da criptografia de chave pública, ficando acessível a sistemas de recursos computacionais limitados.

4. Método de Autenticação

A base do algoritmo de autenticação proposto está na dificuldade de resolver o PLD, quando as variáveis envolvidas são relativamente grandes, este problema se torna computacionalmente inviável, para facilitar o entendimento vamos definir o problema. Dado um p primo e $l, s \in \{1, \ldots, p-1\}$ determinar $s \in \{1, \ldots, p-2\}$ tal que $r \equiv l^s \mod p$. O PLD vem sendo extensivamente aplicado em criptografia assimétrica. O PLD já tem sido usado para autenticação [Tzeng et al. 2007] em um processo de criptografia, no entanto estamos apresentando uma nova perspectiva onde se usa o *login* do usuário no processo de autenticação. Observe que isto é um novo paradigma, uma vez que normalmente não é inviável inserir informações conhecidas como parte de uma chave criptográfica. Em nosso caso veremos que é interessante inserir o *login* no processo.

Daqui em diante denotaremos a variável L como sendo o *login*, ou seja, um número natural mapeado na tabela ASCII e denotaremos por S a senha mapeada da mesma forma. Agora iremos discutir o método de autenticação proposto neste trabalho, com havíamos dito L e S serão dois números naturais, primeiro escolhemos um primo p relativamente grande logo após computamos

$$L^S \mod p,$$

essa será nossa *string* de autenticação. A questão a ser pensada é a probabilidade de colisão, ou seja, dado L_1, L_2, S_1 e S_2, dois pares de *login* e senha respectivamente, em quais circunstâncias teremos

$$L_1^{S_1} \equiv L_2^{S_2} \mod p,$$

sabemos que cada L_i e S_i após serem mapeadas devem pertencer a $\mathbb{Z}_p$ com $L_i \neq 0 \, \forall i \in \mathbb{N}$.

Poderíamos, de alguma forma, retirar o caso em que $S = 1$, isto faria com que o número de colisões diminuísse, deixando esta técnica consideravelmente mais acessível, pois excluiríamos o caso trivial de colisão, onde

$$a^k = (a^k)^1$$

ou seja, $L_1 = a, S_1 = k, L_2 = a^k, S_2 = 1$. Num sistema qualquer não é tarefa difícil aplicar esta restrição. Entretanto temos muitas outras colisões, por exemplo

$$6^2 \equiv 3 \equiv 5^7 \mod 11.$$

5. Método sem colisões

Ao invés de usarmos o *login* L como base vamos usar um elemento g, tal que g seja gerador de um grupo cíclico. Neste caso $g^S \mod p$ gera univocamente todos os elementos de $\mathbb{Z}_p^*$ [Shokranian et al. 1999]. Por simplicidade, considere o seguinte exemplo com números pequenos: faça $g = 6$ e calcule $6^i \mod 11$ para i variando de 1 a 10, logo teremos os números $\{6, 3, 7, 9, 10, 5, 8, 4, 2, 1\} = \mathbb{Z}_{11}^*$. Desta forma não temos colisões e cada senha é direcionada a um número que representa uma *string*, que será armazenada na criação de uma conta de acesso e será comparada na tentativa da conexão. Porém se dois usuários tiverem a mesma senha, ambos terão a mesma *string* e se por acaso um deles tiver acesso a tabela de *strings* poderá saber que outro usuário tem a mesma senha que ele. Para solucionar este problema, considere uma operação de concatenação $\sqcup$, logo tempos que $n = S \sqcup L$, de forma que podemos calcular $g^n \mod p$. Como o L é exclusivo para cada usuário, poderíamos concluir que n também é exclusivo. Observe que existe a possibilidade de um *login* ter uma letra a menos que outro e suas respectivas senhas completaram a concatenação de forma que $S \sqcup L = n = S' \sqcup L'$. Por exemplo, imagine um *login* borges e senha abacaxi, podemos ter um *login* borg e senha esabacaxi. Para garantir a exclusividade de n basta inserirmos um caracter c especial, que não pode ser digitado, entre o *login* e a senha

$$n = S \sqcup c \sqcup L,$$

garantindo que L e S geram valores diferentes para n, se $n < p$.

Com n exclusivo temos uma propriedade interessante do ponto de vista da segurança, não é necessário armazenar os *logins* nas máquinas onde serão feitas as autenticações. Pois $g^n \mod p$ sempre vai geram um valor diferente para cada valor de n, basta comparar se n está em uma lista de autorizados.

6. Segurança

Nesta seção apresentaremos a complexidade para resolução do PLD. Até a atualidade não se conhece um algoritmo de tempo polinomial para resolver este problema. Existe uma vasta literatura que avalia o PLD [Odlyzko 2000, Semaev 1998] sob várias formas. O algoritmo mais rápido para o PLD sobre curvas elípticas possui tempo de execução $O(\sqrt{n})$ [Gordon and McCurley 1993], onde n é a ordem da curva [Teske 1998, Kuhn and Struik 2001]. Observe que este algoritmo é exponencial em relação ao número

de *bits* necessários para representar a ordem do grupo. Para o PLD e o problema da fatoração o tempo de execução é

$$O(\exp(c + O(1))(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}})$$

onde c é uma constante e $n = pq$ [Terada 2000], neste caso a complexidade é considerada sub-exponencial. Logo, é mais difícil de se resolver o PLD sobre curvas elípticas do que o PLD original. Por estes motivos, um criptossistema baseado em curvas elípticas necessita de uma chave de aproximadamente 160 *bits*, e o RSA, por exemplo, utiliza uma chave de 1024 *bits*, mantendo, segurança equivalente em ambos.

Toda a segurança deste método é baseada no PLD, isto é, nas propriedades da potência em grupos cíclicos. Calcular a potência é computacionalmente fácil, mas calcular a função inversa, o expoente, é computacionalmente intratável. A propriedade importante de um grupo cíclico $(G, \cdot)$ é a existência do automorfismo de grupos $G \rightarrow \mathrm{Aut}(G)$ estabelecida pela potência $exp_g : i \mapsto g^i$. Devemos ter cuidado com curvas elípticas, pois nem todo grupo é cíclico.

7. Conclusões

Visto que estamos nos baseados em um problema que é computacionalmente intratável e não se é conhecido um algoritmo que calcule a tal que $\alpha^a \mod p$, dados α e p em tempo polinomial, logo a garantia da segurança do esquema é verificada. É importante ressaltar que nosso método não é uma função de *hash*, por isto estamos livres de colisões, se nossas restrições forem satisfeitas. Nosso método pode ser modelado conforme a necessidade do sistema dando a liberdade de não ter o *login* registrado na máquina de acesso. Uma grande vantagem deste método é a fácil implementação em sistemas, uma vez que não se precisa armazenar o *login* de cada usuário e a aritmética envolvida é relativamente simples. Já as funções de *hash*, na maioria das vezes, transformam uma string de tamanho indefinido em uma de tamanho fixo, permitindo que sejam muito bem definidas as chances de obtermos uma colisão.

8. Agradecimentos

Gostaríamos de agradecer ao CNPq, pelo apoio financeiro para este trabalho.

Referências

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654.

Dobbertin, H. (1996). The status of MD5 after a recent attack. *j-CRYPTOBYTES*, 2(2):1, 3–6.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31(4):469–472.

Fernandes, S. and Reddy, K. L. M. (2002). Securing applications on linux with pam. *Linux J.*, 2002(102):1.

Gordon, D. M. and McCurley, K. S. (1993). Massively parallel computation of discrete logarithms. *Lecture Notes in Computer Science*, 740:312–323.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209.

Koblitz, N. and Menezes, A. J. (2004). A survey of public-key cryptosystems. *SIAM Rev.*, 46(4):599–634 (electronic).

Kuhn, F. and Struik, R. (2001). Extensions of pollard's rho algorithm for computing multiple discrete logarithms. In *SAC '01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, pages 212–229, London, UK. Springer-Verlag.

Liang, J. and Lai, X. (2005). Improved collision attack on hash function md5. Cryptology ePrint Archive, Report 2005/425. `http://eprint.iacr.org/`.

Miller, V. S. (1986). Use of elliptic curves in cryptography. In *Advances in cryptology—CRYPTO '85 (Santa Barbara, Calif., 1985)*, volume 218 of *Lecture Notes in Comput. Sci.*, pages 417–426. Springer, Berlin.

NIST (2002). Secure hash standard. Federal Information Processing Standards Publication 180-2.

Odlyzko, A. (2000). Discrete logarithms: The past and the future. *Designs, Codes, and Cryptography*, 19(2–3):129–145.

Rivest, R. (1992). The md5 message-digest algorithm. *RFC Editor - IETF*, RFC1321.

Semaev, I. A. (1998). An algorithm for evaluation of discrete logarithms in some nonprime finite fields. *Math. Comput.*, 67(224):1679–1689.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423, 623–656.

Shokranian, S., Soares, M., and Godinho, H. (1999). *Teoria dos números*. Editora Universidade de Brasilia.

Stevens, M. (2006). Fast collision attack on md5. Cryptology ePrint Archive, Report 2006/104. `http://eprint.iacr.org/`.

Terada, R. (2000). *Segurança de Dados: Criptografia em Redes de Computadores*. Edgard Blucher, 1 edition.

Teske, E. (1998). Speeding up pollard's rho method for computing discrete logarithms. In *ANTS-III: Proceedings of the Third International Symposium on Algorithmic Number Theory*, pages 541–554, London, UK. Springer-Verlag.

Tzeng, S.-F., Tang, Y.-L., and Hwang, M.-S. (2007). A new convertible authenticated encryption scheme with message linkages. *Comput. Electr. Eng.*, 33(2):133–138.

Wang, X., Yin, Y. L., and Yu, H. (2005). Finding collisions in the full sha-1. In *CRYPTO*, pages 17–36.

Wang, X. and Yu, H. (2005). How to break md5 and other hash functions. In *EUROCRYPT*, pages 19–35.