鍵共有プロトコルとパケット制御に関する一考察
A Study on Packet Control in Key-Agreement Protocols

松浦 幹太*　　今井 秀樹*
Kanta Matsuura　　Hideki Imai

Abstract—Depending on the protocol design, Denial-of-Service (DoS) attacks may threaten key-agreement protocols. Since verification cost might be abused for DoS, authentication algorithms/mechanisms should be designed in a DoS-resistant way. In addition, DoS-resistance may depend on packet-control mechanisms, too. This paper studies the effect of random packet discarding. The results show (1) that simple random discarding improves the blocking probability (BP) insignificantly in interested region such as BP>1% and (2) that performance reduction for innocent initiators are rather significant. This trade-off can be improved by using more sophisticated discarding policies.

Keywords— key-agreement protocol, authentication, Denial-of-Service attack, packet control.

1　はじめに

鍵共有プロトコルでは、ふつう、正しい相手とやりとりしていることを確認する必要がある。そこで、公開鍵を利用して何らかの認証がなされることが多いが、その結果としてプロトコルにべき乗剰余演算のような比較的負荷の高い計算プロセスが付加される。そのため、不正な接続要求者がでたらめな要求を送り続ければ、応答者はそれらの認証作業に多大な計算機資源を浪費せねばならず、ついにはシステムダウンを招く恐れがある。これはサービス妨害 (DoS: Denial-of-Service) 攻撃の 1 種であり、セキュリティ機能の導入そのものを脅かす大きな問題である。

このような CPU 枯渇型の攻撃対策として攻撃者にも計算機資源の浪費を強制することが提案され [1], [2]、サーバが利用可能になる閉塞率を計算することによって妨害耐性が評価されている [3], [4]。本稿では、3 回のメッセージ授受から成るプロトコルにランダムパケット棄却という制御方式を加えた場合に閉塞率が受ける影響を考察する。

2　サービス妨害攻撃対策

CPU 枯渇対策として提案された基本方針は以下の通りである。プロトコル構成は、最初に始動者が要求メッセージを送り、応答者が応答メッセージを返し、それに対してさらに始動者が確認メッセージを応答者へ送り返すという 3 パスのものを想定している。

1. べき乗剰余演算のように負荷の高い計算を、要求メッセージ受付直後に行わない。
2. 応答メッセージにランダムなパラメータを暗に含ませ、それを復元するためには始動者がべき乗剰余演算のように負荷の高い計算をしなければならないようにする。
3. 前項のランダムパラメータを認証に利用する。
4. 復元したランダムパラメータを用いなければ生成できないパラメータ（ここでは「確認パラメータ」と呼ぶ）を、確認メッセージに含める。
5. 確認パラメータの生成に正しいランダムパラメータが使われたかどうかを、確認メッセージ受付直後にハッシュ関数などの負荷の低い計算で確認できるようにする。

以上の方針にしたがった Diffie-Hellman 型の鍵共有プロトコルの例を、図 1に示す。応答者が行う負荷の高い計算で始動者に依存するもの（したがって事前計算不可能なもの）は、「始動者がプロトコルにしたがって（負荷の高い計算で）応答者を認証した」ことが確認されてから実行される。この結果、攻撃者は被攻撃者（応答者）と同レベルの CPU 資源を浪費しなければ、サービス妨害を行うことができない。サービス妨害攻撃対策としては、Cookie を利用して攻撃者の IP アドレス等のセッション情報も認証に用いるものもある [5]。これには、攻撃者が踏み台を占拠してそこから不正なリクエストを発する場合に弱いという欠点がある。それに対し、攻撃者に負荷を強いる方式では、踏み台にもある程度対処できる。

図 1のプロトコルは「応答者がハングアップするか否かは、CPU ではなくメモリ枯渇で決定される」と仮定し、図 2のような状態遷移図における統計的平衡状態を求めて評価できる。

一方、鍵共有プロトコルに依存しない DoS 対策として、正当か不正かに関わらず応答者側でランダムにリクエストを棄却するという方法も考えられる。その問題点の一つは、正当な利用者の利便性とのバランスをとるの

---

* 東京大学生産技術研究所. 〒 106-8558 港区六本木 7-22-1. Institute of Industrial Science, University of Tokyo, Roppongi 7-22-1, Minato-ku, Tokyo 106-8558, JAPAN. E-mail: kanta@iis.u-tokyo.ac.jp, imai@iis.u-tokyo.ac.jp

が難しいことである。しかも、バランスをとる以前に、ランダム棄却の効果に関する定量的評価が不足している。次章では、図 2のモデルを応用した定量的評価を報告する。

3 パケット棄却の影響

応答者側がランダムにリクエストを棄却する確率を α=1.0% とする。この時、図 2における攻撃規模 Mを $(1-\alpha)M$で置き換え、要求生起率 Rを $(1-\alpha)R$で置き換えてサーバ閉塞率を求めた。その結果を図 3に ◦ 印でプロットした。比較のため、ランダム棄却を行わない場合の閉塞率を図 3に × 印でプロットした。これらは、図 3の下方のカーブでほとんど重なっている。

ランダム棄却する場合、サービス妨害攻撃でない正常なリクエストも、棄却されてしまう可能性がある。すなわち、

(a) 事前計算値が枯渇して閉塞している時に到来した場合

または

(b) 閉塞してはいないがランダム棄却に掛かったため受け付けられなかった場合

に棄却されてしまう。その確率を同じく図 3に $*$ 印でプロットした。(b) のパターンで連続して棄却される確率は累乗的に減少するので問題ないと解釈することもできるが、1 回正常なリクエストを発してそれが受け付けられない確率は、図 3の上方の曲線のような特性で与えられる。その結果生じる再送制御等の手間をかけて、下方の曲線の × 印から ◦ 印程度の閉塞率改善を希望するかどうかどうかを考えれば、ランダムなパケット棄却はあまり望ましい制御方式ではないといえる。

4 むすび

CPU 枯渇対策を施して閉塞を事前計算値の有無で判断できるようになれば、定量的な DoS 耐性評価が容易になる。本稿では、そこで用いる評価モデルにおいてランダムなパケット棄却の効果を考慮する方法を述べ、実際に数百 KB 程度のメモリを仮定して評価した。その結果、毎分 50000 個から 80000 個程度の不正な要求に襲われる場合、ランダムなパケット棄却の効果がほとんどないことを明らかにした。今後は、ランダムでない適応的なパケット制御方式の影響や、閉塞に至る前の中間的な状態におけるパフォーマンス低下の問題等も考察したい。

参考文献

[1] 廣瀬勝一, 松浦幹太. サービス拒否攻撃に対して耐性のある鍵共有プロトコル. 電子情報通信学会 1998 年基礎・境界ソサイエティ大会講演論文集, A–7–7.

[2] 松浦幹太, 今井秀樹. 鍵共有プロトコルのサービス妨害攻撃対策. 第 21 回情報理論とその応用シンポジウム予稿集, pp.719-722, December 1998.

[3] K. Matsuura and H. Imai. "Resolution of ISAKMP/Oakley Key-Agreement Protocol Resistant against Denial-of-Service Attack". In *Pre-Proc. of Internet Workshop'99 (IWS'99)*, pp. 17–24, February 1999.

[4] Kanta Matsuura and Hideki Imai. "Protection of Authenticated Key-Agreement Protocol against a Denial-of-Service Attack". *Cientifica Journal of The National Polytechnic Institute of Mexico* (to appear).

[5] P. Karn and W. Simpson. "Photuris: Session-Key Management Protocol". rfc2522. March 1999.

[6] M. Bellare, R. Canetti, and H. Krawczyk. "Keying hash functions for message authentication". In *Advances in Cryptology — CRYPTO'96*, pp. 1–15, August 1996. Springer-Verlag. Lecture Notes in Computer Science 1109.

Step	**始動者 (Initiator)** 秘密鍵: $SK_I \in_R [1, 2, \cdots, q-2]$ 公開鍵: $PK_I = g^{SK_I} \bmod p$		**応答者 (Responder)** 秘密鍵: $SK_R \in_R [1, 2, \cdots, q-2]$ 公開鍵: $PK_R = g^{SK_R} \bmod p$
((1))	$k_I, r_I \in_R [1, 2, \cdots, q-2]$ $u_I = g^{k_I} \bmod p$ $s_I = g^{r_I} \bmod p$		$k_R, r_R \in_R [1, 2, \cdots, q-2]$ $u_R = g^{k_R} \bmod p$ $s_R = g^{r_R} \bmod p$
((2))	Construct *etc*.		
((3))	$\{u_{\mathrm{I}}, ID_{\mathrm{I}}, etc\}$	$\Rightarrow$	
((4))			Reshape *etc* into $etc^{\dagger}$. $e_R = hash\left(s_R, u_R, u_I, ID_R, ID_I, etc^{\dagger}\right)$ $w_R = r_R / \left(k_R + e_R SK_R\right) \bmod q$
((5))		$\Leftarrow$	$\{u_R, e_R, w_R, ID_R, etc^{\dagger}\}$
((6))	$\tilde{s}_R = u_R^{w_R} PK_R^{e_R \cdot w_R} \bmod p$ Check $e_R = hash\left(\tilde{s}_R, u_R, u_I, ID_R, ID_I, etc^{\dagger}\right)$? If successful, $H = hash\left(\tilde{s}_R, u_R, u_I\right)$ $K_{IR} = u_R^{k_I} \bmod p$ Reshape *etc* and $etc^{\dagger}$ into $etc^{\ddagger}$. $z_I = KH_{K_{IR}}\left(etc^{\ddagger}\right)$ $e_I = hash\left(s_I, u_I, u_R, ID_I, ID_R, etc^{\dagger}, z_I\right)$ $w_I = r_I / \left(k_I + e_I SK_I\right) \bmod q$		
((7))	$\{H, e_I, w_I\}$	$\Rightarrow$	
((8))			Check $H = hash\left(s_R, u_R, u_I\right)$? If successful, $\tilde{s}_I = u_I^{w_I} PK_I^{e_I \cdot w_I} \bmod p$ $K_{RI} = u_I^{k_R} \bmod p$ Reshape *etc* and $etc^{\dagger}$ into $etc^{\ddagger}$. $z_R = KH_{K_{RI}}\left(etc^{\ddagger}\right)$ Check $e_I = hash\left(\tilde{s}_I, u_I, u_R, ID_I, ID_R, etc^{\dagger}, z_R\right)$?
((9))	Accept K_{IR} as the session key.		Accept K_{RI} as the session key.

図 1: サービス妨害攻撃対策を施した改良型 Diffie-Hellman 鍵共有プロトコルの例. p は大きな素数, q は $p-1$ の大きな素因数, g は $Z_p^* = \{1, 2, \cdots, p-1\}$ における位数 q の元である. ID_I と ID_R は, それぞれ始動者と応答者の ID である. *etc*, $etc^{\dagger}$, $etc^{\ddagger}$ はプロトコルバージョンやオプション記述子, 補助的な乱数 (nonce), クッキー (Cookie) など, 実際のプロトコル実装で組み込まれるパラメータを表す. $\tilde{s}_I$ と $\tilde{s}_R$ は, それぞれ s_I と s_R を復元した値である. $hash(\cdot)$ は一方向性ハッシュ関数, $KH_k(\cdot)$ は鍵 (k) 付きハッシュ関数 [6] である.

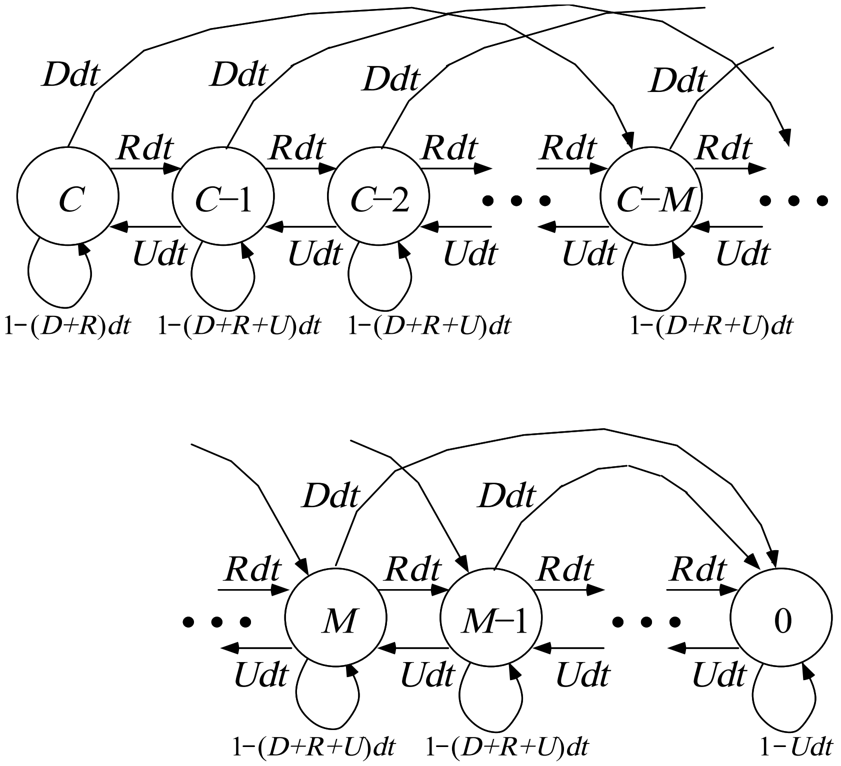

図 2: 鍵共有プロトコル応答者の状態遷移図．各状態は，応答者が保持している事前計算値 u_R, s_R の組数 (最大で C) で表す．攻撃者が 1 回の攻撃で多数 (M個とする) の不正な要求を送りつけると，状態は Mだけ低くなる．時域 $[t, t+dt]$ の間に 1 塊の攻撃が到来する確率は dt に比例し，$D \cdot dt$ である．同じく正当なリクエストが到来する確率も dt に比例し，$R \cdot dt$ である．新たな事前計算値を 1 組計算できる確率も dt に比例し，$U \cdot dt$ である．

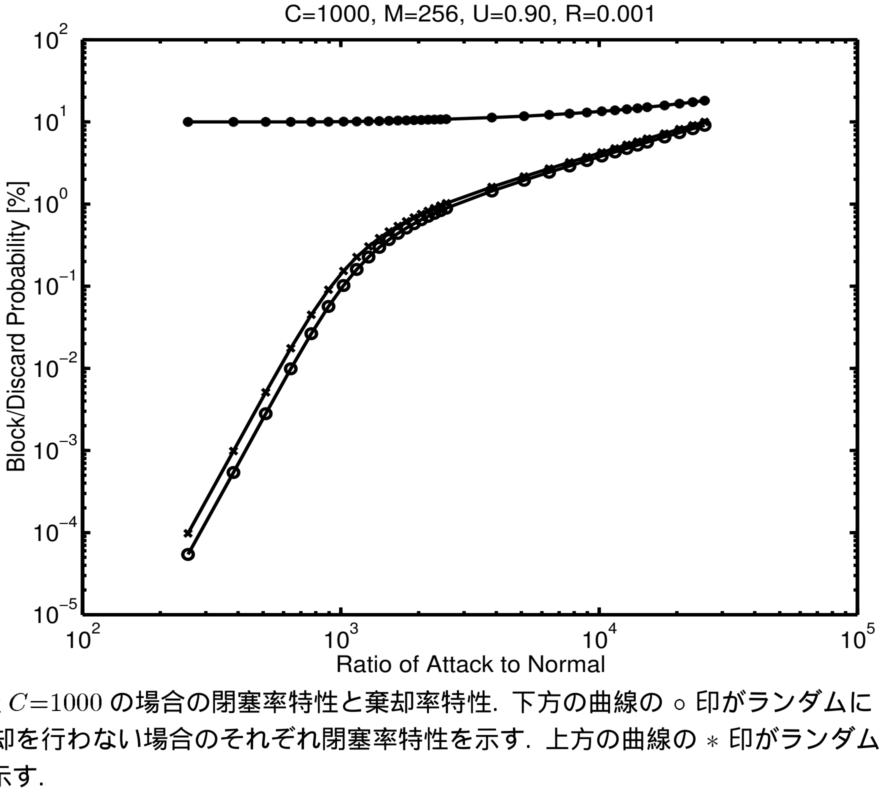

図 3: 事前計算容量 $C{=}1000$ の場合の閉塞率特性と棄却率特性．下方の曲線の $\circ$ 印がランダムに 10% 棄却した場合，$\times$ 印がランダム棄却を行わない場合のそれぞれ閉塞率特性を示す．上方の曲線の $*$ 印がランダムに 10% 棄却した場合の棄却率特性を示す．