認証と鍵共有プロトコル

情報セキュリティ大学院大学　有田正剛

平成 19 年 6 月 11 日

目 次

1　はじめに

今回は鍵共有プロトコルについてお話します。鍵共有プロトコルとはセッション鍵を共有するためのプロトコルです。セッション鍵 k とは一時的な使い捨ての秘密の鍵情報を意味し、鍵共有プロトコルを実行した 2 者 P_1, P_2 の間でだけ共有されて、その後の P_1, P_2 の通信を秘匿したり (例えば k を鍵としてブロック暗号を用いる)、また認証したり (例えば k を鍵として認証子を用いる) するために使われます。

ここでは、(数多ある) 鍵共有プロトコルの中から、2 つの典型的なプロトコルを選んで紹介します。そのうちの 1 つは公開鍵暗号を用いる鍵共有

法で、SSL にも用いられています。もう 1 つは DH 鍵共有法と呼ばれるもので IPSec に用いられています。

それら 2 つのプロトコルを縦糸にし、横糸として鍵共有プロトコルの安全性を考えます。鍵共有プロトコルの安全性は、一見すると単純に見えます。セッション鍵 k を守ればいいわけなので、「プロトコルを実行している P_1, P_2 のメッセージのやり取りを敵 A が覗き見しても、敵 A にはセッション鍵 k がわからない」こととしておけば十分に思えます。

ところが、鍵共有プロトコルはそれ自身では意味を持たず、他のプロトコルやシステム（「親プロトコル」と呼びます）で部品として用いられて初めて意味をもつプロトコルです。しかも、その親プロトコルは、1 つだけでなく、同時に多数の鍵共有プロトコルを部品として実行することも珍しくありません。そのため、鍵共有プロトコルは、同時に複数のコピーがいくらかの文脈を共有した状態で実行されます。このような状態での実行を「コンカレントな」実行と呼びますが、このコンカレントな実行において安全性を達成するのはなかなか困難なことです。鍵共有プロトコルがどのようにしてコンカレントな実行に対処しているかを見たいと思います。

2　鍵共有プロトコルとは?

鍵共有プロトコルとはセッション鍵を共有するための 2 者間プロトコルです。鍵共有プロトコルを実行する、2 つのパーティ P_1 と P_2 は、いくつかのメッセージを交換し、それら交換したメッセージをもとにして、それぞれ、あるビット列 k を計算して出力します。ここで、P_1 も P_2 も同じ k を出力できなければなりません。この k をセッション鍵と呼びます。

このセッション鍵 k は一時的な使い捨ての、P_1 と P_2 の間だけの秘密情報を意図したもので、通常、その後の P_1 と P_2 間の通信を秘匿したり (例えば k を鍵としてブロック暗号を用いる)、また認証したりする (例えば k を鍵として認証子を用いる) ために使われます。

セッション鍵 k は鍵共有プロトコルを実行した 2 者 P_1, P_2 だけが知るべきなので、鍵共有プロトコルに以下の安全性条件 SK を求めるのは自然です。

安全性条件 SK　鍵共有プロトコルを実行する 2 者 P_1, P_2 のやり取りするメッセージをすべて敵 A が覗き見しても、敵 A にはセッション鍵 k のどのような部分情報もわからない。

ここで、単に敵 A にはセッション鍵 k がわからないだけでなく、セッション鍵 k のどのような部分情報（例えば k の第 3 ビット目 k_3 が 1 であるとか、$k_1 \oplus k_7 = 0$ であるとか) もわからないことを要求していること

に注意してください。このことが必要なのは、例えば、この k を鍵にしてブロック暗号でメッセージを暗号化して秘匿通信するような場合を考えると明らかです。ブロック暗号の安全性は鍵が完全に秘密であるときにしか保証されません。

3　2 つの鍵共有プロトコル：基本形

2 つの鍵共有プロトコル（の基本形）を紹介します。1 つは公開鍵暗号ベースの鍵共有プロトコル EB、もう 1 つは Diffie-Hellman 鍵共有プロトコル DH です。

3.1　公開鍵暗号ベースの鍵共有プロトコル EB

用いる公開鍵暗号を $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ とし (G:鍵生成、E:暗号化、D:復号アルゴリズム)、n をセキュリティパラメータ（安全性を指定するパラメータ）とします。P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得ます。

1° P_2 は、G を実行して鍵ペア (e, d) を生成する: $(e, d) \leftarrow \mathsf{G}(n)$. 公開鍵 e を P_1 に送る。

2° P_1 は、ランダムな n ビットのビット列 k を生成し、それを鍵 e で暗号化して暗号文 c を得る：$k \xleftarrow{\$} \{0,1\}^n$, $c \leftarrow \mathsf{E}_e(k)$. c を P_2 に送り、k を出力する。

3° P_2 は、c を復号して k を得る: $k \leftarrow \mathsf{D}_d(c)$. k を出力する。

プロトコルを実行するたびに鍵生成 $(e, d) \leftarrow \mathsf{G}(n)$ を実行するのは非効率なので、予め鍵を計算しておき、保存しておいた (同じ)e を送る（あるいはディレクトリに公開しておき、P_1 はそれから e を入手する）ことにしても OK です。

敵 A が P_1, P_2 間でやり取りされるメッセージ e と c を入手しても、公開鍵暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ の安全性より (c に暗号化されている)k は分かりません。ただし、安全性条件 SK を満たすためには、k のどのような部分情報も A にはわからない必要があるため、公開鍵暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ は強秘匿 (第 1 講「公開鍵暗号入門」参照) である必要があります。

SSL では上の公開鍵暗号として RSA-PKCS#1 が使われています。

3.2　Diffie-Hellman 鍵共有プロトコル DH

p を十分大きな素数とし、$g(\in \{0, 1, \ldots, p-1\})$ を、p を法とする位数が素数 q である整数 (すなわち、g は q 乗して初めて (p でわったあまりが)1

となる数) とします。P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得ます。

1° P_1 は、$\{1, 2, \ldots, q\}$ の範囲からランダムに x を選ぶ: $x \overset{\$}{\leftarrow} \{1, 2, \ldots, q\}$. g の x 乗を計算し X とする: $X = g^x \bmod p$. X を P_2 に送る。

2° P_2 は、$\{1, 2, \ldots, q\}$ の範囲からランダムに y を選ぶ: $y \overset{\$}{\leftarrow} \{1, 2, \ldots, q\}$. g の y 乗を計算し Y とする: $Y = g^y \bmod p$. Y を P_1 に送る。

3° P_1 は、$k = Y^x$ を計算し、k を出力する。

4° P_2 は、$k = X^y$ を計算し、k を出力する。

ここで、まず、

$$Y^x = g^{xy} = X^y$$

であることに注意します。これより、P_1 と P_2 は確かに同じ k を出力することが分かります。

安全性については、数論的な仮定が必要となります。

CDH 仮定 $\{1, 2, \ldots, q\}$ の範囲からランダムに a, b を選ぶとき、どのような効率的なアルゴリズムも、g^a と g^b を与えられても g^{ab} はわからない。

DDH 仮定 $\{1, 2, \ldots, q\}$ の範囲からランダムに a, b, c を選ぶとき、どのような効率的なアルゴリズムも、g^a と g^b を与えられて g^{ab} と g^c を区別（することすら）できない。

DDH 仮定のもとで、鍵共有プロトコル DH は安全性条件 SK を満たします。P_1, P_2 間でやり取りされるメッセージ $X = g^x$,$Y = g^y$ からセッション鍵 $k = g^{xy}$ を求めることは、まさに CDH および DDH 仮定で問題とされていることと同じだからです。

4 コンカレントな環境における鍵共有プロトコルの安全性

ここで、鍵共有プロトコルの本来の目的に戻ります。鍵共有プロトコルはそれ自身では意味がありません。ただ k を共有しても使われないなら意味が分かりません。鍵共有プロトコルは、他のプロトコルやシステム（「親プロトコル」と呼びます）で部品として用いられて初めて意味をもつプロトコルです。

しかも、その親プロトコルは、1 つだけでなく、同時に多数の鍵共有プロトコルを部品として実行することも珍しくありません。そのため、鍵共有プロトコルは、同時に複数のコピーが (いくらかの文脈を共有した状態で) 実行されます。しかも、敵 A はただメッセージを覗き見るだけとも限りません。敵は、同時に実行されている数多くの鍵共有プロトコルの参加者としても潜んでいる可能性があります（正当な参加者として参加している場合もあれば、なりすましている場合もある)。さらに、敵 A はすでに終了した過去の鍵共有プロトコルに関しては、そのセッション鍵 k を入手できる場合もあります（鍵共有プロトコルを利用するアプリケーションがセッション鍵をばらしてしまう場合など)。このような鍵共有プロトコルの、より現実に近い、実行環境をコンカレントな環境と呼びます。

コンカレントな環境にも耐えられる、より強力な安全性条件 CSK を設定します。

安全性条件 CSK コンカレントな実行環境、すなわち、

(設定 1) 鍵共有プロトコルは、同時に複数のコピーが (いくらかの文脈を共有した状態で) 実行される、

(設定 2) しかも、敵 A 自身もそれらの任意の鍵共有プロトコルの実行に（正当な参加者としてあるいは誰かに成りすまして）参加できる、

(設定 3) さらに、敵 A はすでに終了した過去の任意の鍵共有プロトコルを選択し、そのセッション鍵を入手できる

環境において、

(条件 1) 鍵共有プロトコルの実行において、P_1 と P_2 さえ正しく正直に振舞えば、(陰日向で敵 A がどのように振舞おうとも) P_1 と P_2 の間で正しくセッション鍵が共有され、かつ

(条件 2) (敵 A の参加していない) 鍵共有プロトコル実行の全てのメッセージを敵 A が覗き見しても、(設定 3 の手段を用いてセッション鍵を入手しない限り) 敵 A にはセッション鍵 k のどのような部分情報もわからない。

4.1 プロトコル DH に対する中間者攻撃

3.2 節の DH 鍵共有プロトコル DH は安全性条件 CSK を満たしません。実際、コンカレント環境においては、以下のような中間者攻撃が成立します。

1° P_1 は正直に $X(=g^x)$ を計算し、P_2 になりすました攻撃者 M に送る。

2° M は $X'(=g^{x'})$ を自身で計算し、P_1 になりすまして P_2 に送る。

3° P_2 は正直に $Y(=g^y)$ を計算し、P_1 になりすました攻撃者 M に送る。

4° M は $Y'(=g^{y'})$ を自身で計算し、P_2 になりすまして P_1 に送る。

以上の実行は、P_1 と P_2 の視点からは全く正常に見えていることに注意します。ところが、P_1 は $k_1 = g^{xy'}$ を、P_2 は $k_2 = g^{x'y}$ を出力し、この k_1, k_2 は異なるので、安全性条件 CSK の条件 1 が破られています。さらに悪いことに M は上の k_1 も k_2 も知っています。

4.2 プロトコル EB に対する再送攻撃

3.1 節の公開鍵暗号ベースの鍵共有プロトコル EB は安全性条件 CSK を満たしません。実際、コンカレント環境においては、以下のような再送攻撃が成立します。ここでは、プロトコル EB において P_1 は予め P_2 の公開鍵 e を入手しているとします。(したがって、プロトコル EB で送受信されるメッセージは P_1 から P_2 へ送られる暗号文 c のみとなります。)

1° P_1 と P_2 は正直にプロトコル EB を実行する。このとき、P_1 から P_2 へ送られる暗号文 c を敵 R が入手する。

2° 敵 R は正当な送信者として P_2 との間でプロトコル EB を実行する。ただしこの際、上で入手した暗号文 c を P_2 へ送る。

3° 2° の R と P_2 によるプロトコル EB の実行に対し、R は設定 3 の手段を用いてそのセッション鍵 k を入手する。(k は C の復号文。)

4° 敵 R は 1° で P_1 と P_2 が得たセッション鍵は k であることを知る。

敵 R は覗き見したメッセージを自分自身のメッセージとして再送することで安全性条件 CSK の条件 2 を破ってしまいました。

この攻撃において、P_2 は同じ暗号文 c を異なる送信者 P_1 と R から 2 度受信しているので、R からの攻撃に気づくのではないか、と思われるかもしれません。しかし、P_2 は人ではなく、単にアルゴリズムなので、予めコードとして書かれた手続きしか実行しません。したがって、受信した c が再送されたものかどうかのチェックはしないため、R からの攻撃を検知しません。

5　2つの鍵共有プロトコル：発展形

ここでは、2つの鍵共有プロトコル EB と DH をコンカレント環境に耐えられるよう、強化します。強化の方針は、以下の通りです。

プロトコルの各メッセージにセッション識別子を含める。
プロトコル一つ一つの実行をセッションと呼びます。コンカレント環境では同時に複数のセッションが発生するので、それらの異なるセッションに属するメッセージが混じってしまわないよう、各メッセージにセッション識別子を含めます。

プロトコルの各メッセージに送信者識別子を含める。
異なる送信者が生成したメッセージが混じってしまわないよう、各メッセージに送信者識別子を含めます。

プロトコルの各メッセージの送信に認証メカニズムを組み込む。
折角、セッション識別子や送信者識別子をメッセージに含めても、それを敵に悪用されては意味がなくなります。そのようなことを防ぐため、各メッセージの送信にチャレンジレスポンスに基づく認証メカニズムを組み込みます。具体的には、送信者 P_1 が受信者 P_2 にメッセージ m を送るところを、各パーティはデジタル署名スキームを利用できるものとして、以下のように変更します。まず受信者 P_2 の方から送信者 P_1 に乱数 N を送り、それに対して、送信者 P_1 がメッセージ m に m と N の連結文 (m, N) の署名文 s を添付して送るようにします。これによって、m は確かに今現在の P_1 が送ったことを受信者 P_2 は確信することができます。

5.1　鍵共有プロトコル DH-SIG

各パーティはデジタル署名スキームが使用できるものとし、P_i のメッセージ m に対する署名文を $SIG_i(m)$ で表します。各パーティ P_i の識別子を (記号を節約して)P_i と書きます。セッション識別子を s とします。セッション識別子 s は親プロトコルから与えられると考えます。

P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得ます。

1° P_1 は、$\{1, 2, \ldots, q\}$ の範囲からランダムに x を選ぶ: $x \xleftarrow{\$} \{1, 2, \ldots, q\}$.
g の x 乗を計算し X とする: $X = g^x \bmod p$. (P_1, s, X) を P_2 に送る。

2° P_2 は、$\{1, 2, \ldots, q\}$ の範囲からランダムに y を選ぶ: $y \xleftarrow{\$} \{1, 2, \ldots, q\}$.
g の y 乗を計算し Y とする: $Y = g^y \bmod p$. そして P_2 は、$(P_2, s, Y, SIG_2(P_2, s, Y, X, P_1))$ を P_2 に送る。

3° P_1 は、署名文 $SIG_2(P_2, s, Y, X, P_1)$ の正しさを確認する。(もし正しくなければプロトコル実行を拒否する。) そして P_1 は、$(P_1, s, SIG_1(P_1, s, X, Y, P_2))$ を P_2 に送ると同時に、$k = Y^x$ を計算し、(s, k) を出力する。

4° P_2 は、署名文 $SIG_1(P_1, s, X, Y, P_2)$ の正しさを確認する。(もし正しくなければプロトコル実行を拒否する。) P_2 は、$k = X^y$ を計算し、(s, k) を出力する。

鍵共有プロトコル DH-SIG には、プロトコル SIG に対して成立した中間者攻撃 (4.1 節) は通用しません。DH-SIG ではメッセージが (ランダムチャレンジに対応する) 署名文で守られているため、敵 M が P_1 や P_2 になりすましてメッセージを送ることができないためです。例えば署名文 $SIG_2(P_2, s, Y, X, P_1)$ には送信者と受信者の識別子が含まれていることに注意してください。

5.2 鍵共有プロトコル EB-SIG

各パーティはデジタル署名スキームが使用できるものとし、P_i のメッセージ m に対する署名文を $SIG_i(m)$ で表します。各パーティ P_i の識別子を (記号を節約して)P_i と書きます。セッション識別子を s とします。セッション識別子 s は親プロトコルから与えられると考えます。さらに、$f_\kappa(\cdot)$ を擬似ランダム関数 (κ を知らないものにはランダム関数と区別がつかない関数) とします。

P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得ます。(各パーティは予め必要な公開鍵と検証鍵を入手しているとします。)

1° P_2 は、ランダムな n ビットのビット列 N を生成し、(P_2, s, N) を P_1 に送る。

2° P_1 は、ランダムな n ビットのビット列 κ を生成し、それを鍵 e で暗号化して暗号文 c を得る：$k \xleftarrow{\$} \{0,1\}^n$, $c \leftarrow \mathsf{E}_e(\kappa)$. そして P_1 は、$(P_1, s, c, SIG_1(P_1, s, c, N, P_2))$ を P_2 に送り、$k = f_\kappa(P_1, P_2, s)$ を出力する。

3° P_2 は、署名文 $SIG_1(P_1, s, c, N, P_2)$ の正しさを確認する。(もし正しくなければプロトコル実行を拒否する。) c を復号して κ を得る: $\kappa \leftarrow \mathsf{D}_d(c)$. $k = f_\kappa(P_1, P_2, s)$ を出力する。

鍵共有プロトコル EB-SIG には、プロトコル EB に対して成立した再送攻撃 (4.2 節) は通用しません。敵 R が先と全く同じようにメッセージ

(P_1, s, c, SIG_1) をまるごと再送しても、新しいランダムチャレンジ N に対応していないので署名がおかしくなります。敵 R が c だけ再利用し、署名文は新しいランダムチャレンジ N に対応して自身で再計算して添付しても、セッション鍵 $k = f_\kappa(P_1, P_2, s)$ は、新しいセッション識別子 s(その他) にリンクしているので、元の値とは違うランダムな値になります。

6 おわりに

コンカレント環境における鍵共有プロトコルのフォーマルな安全性定義がいくつか提案されています。そのもとで鍵共有プロトコル EB-SIG、DH-SIG はその安全性が適切な仮定のもとで証明されています。

ただし、鍵共有プロトコルは、理論と実際が最も鋭く対応するプロトコルといえると思います。そのフォーマルな安全性定義についても、どこまでモデルに現実的な状況を取り入れられているか、また、理論的になりすぎて安全性要件として不自然に厳しくなっていないか、などについて議論が多くあります。