Criptografia: Parte 2

O problema: cifras garantem que a mensagem não foi lida, mas isto não implica que a mensagem não foi alterada.

MAC (Message authentication code)

The MAC authentication assures authentication and integrity, because assures that a message from a A is being sent to B.

But it doesn't assure confidentiality, since we are not encrypting the messages.

The protocol

The computer A generates the message M and generates a tag from the message and a key:

```
t = MAC(key, message)
```

We are assuming that both computers owns the key.

Later, the machine B will receive the message, but it will generate `t'` from a key that he owns and the received message.

```
t' = MAC(key, message)
if (t' != t) return 1;
return 0;
```

Cookies authenticity

Let's suppose that we are playing a game.

The client should not contain the key of the message, since the player would be able to change its own information.

Authenticity in protocols

SSH

The SSH is a type of encryption that uses both ciphers and MAC.

```
m = ENC(Message, key1)
t = MAC(Message, key2)
Send [ m | t]
```

SSL

We can have here malicious code while the program checks if the tag is valid.

```
t = MAC(Message, key2)
mt = ENC([Message | t], key1)
Send mt
```

IPSEC

```
c = ENC(Message, key1)
t = MAC(c, key2)
Sent [c | t]
```

AEAD

(Authentication Encryption with Associated Data).

Usually, we don’t talk about ciphers and MAC separately.

```
Enc(n,k,m,data) -> (c,t) [nounce, key, message and the metadata]
Dec(n,k,c,t) ->  m, only if (c,data) are authentic.
```

> Muito utilizado em **AES-GCM (AES em Galois-Counter-Mode).** É muito eficiente devido ao suporte em HW.

Randomization (aleatoriedade)

Analyzing cryptography we always assume perfect randomization:

- Sequence of bits
- Each bit independent of each other
- There's a probability of 50% for each bit be 1/0.

But in practice this is hard.

Heuristics

Fonts of entropy (really random)

- temperature of the system
- core activity
- user activity

This is slow to generate.

PRG

Contains a state that constantly updated with the entropy fonts.

When its asked to get the next number, the number is generated then the state will be updated.

The problem is that, we might ask for new random numbers, but actually the fonts of entropy haven't change enough.

In linux we have files to generate these numbers: `/dev/urandom` and `/dev/random`.

When to use /dev/random vs /dev/urandom

Use /dev/urandom for most practical purposes. The longer answer depends on the flavour of Unix that you're running. Historically, /dev/random and /dev/urandom were introduced at the same time.

https://unix.stackexchange.com/questions/324209/when-to-use-dev-random-vs-dev-urandom

In `/dev/random` if it gets to the conclusion that it doesn't have enough entropy, then it block. `/dev/urandom` is non-blocking.

We have a trade-off here. We don't really know how good the system tells apart a good entropy and a bad one and also some people don't eve know that `/dev/random` blocks and when this happens it returns a null pointer. For these reason, some applications don't check if `/dev/random/` returns a null pointer and proceed with the process by not providing any randomization.

Cookie poisoning

What Is Cookie Poisoning?

Cookie poisoning is an attack strategy in which the attacker alters, forges, or extracts data from a cookie to steal data or commit fraud.

https://www.f5.com/services/resources/glossary/cookie-poisoning

"Attackers can intercept cookies before they return to the server to extract information from or modify them. Forged cookies can also be created from scratch as a means of impersonating a user to access additional user data. Cookie poisoning is thus a misnomer in that it is often used to refer not only to modified ("poisoned") cookies but to a variety of methods for stealing data from valid cookies or other malicious use of them"

Imagine que temos o cenário de um jogo em que o score é guardado no lado do cliente como uma cookie + tag do protocolo MAC. **O MAC evita com que a pessoa possa alterar a mensagem sem alterar o MAC.** Se a tag do MAC não existisse o jogador poderia simplesmente alterar a cookie e quando o servidor requisitasse este, lia o valor introduzido pelo atacante.

Ainda, precisamos ter um número de sequência, pelo que pode-se enviar a mesma cookie repetidas vezes para o servidor.

HMAC:

`t = H(okey || H(ikey || m))`

Poly1305

Just one time MAC.

Using Poly1305 MAC key multiple times

Thanks for contributing an answer to Cryptography Stack Exchange! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or

https://crypto.stackexchange.com/questions/77886/using-poly1305-mac-key-multiple-times

Perguntas

1) No que consiste o protocolo MAC? Quais são as suas garantias?

▼ Resposta

O MAC (Message Authenticate Code) é um protocolo que garante **autenticidade** e **integridade** de mensagens. Quando um computador (**A**) deseja enviar uma mensagem a outro (**B**), deseja-se que **B** tenha certeza de a mensagem recebida foi enviada por **A**. Para isto, **A** envia a mensagem junto com uma **tag,** gerada a partir de uma chave e da mensagem $t = mac(k, m)$. A chave de **A** é pré partilhada com **B** e quando **B** recebe a mensagem ele calcula uma tag t' da mesma forma que **A.** Se for concretizado que $t' = t$, então podemos dizer que a mensagem veio de **A.**

2) Explique as diferenças entre SSH, SSL e IPSEC.

▼ Resposta

Os três são protocolos que tentam garantir os princípios de autenticidade, integridade e ainda confidencialidade.
SSH → $m' = [enc(m, k_{enc})||mac(m, k_{mac})]$

SSL → $m' = [mac(enc(m, k_{enc}), k_{mac}]$
IPSEC → $x = enc(m, k_{enc}) \rightarrow m' = [x||mac(x, k_{mac})]$

3) O que é AEAD? Como ele é utilizado?

▼ Resposta

É uma maneira de juntar MAC e Cifras.

```
Enc(m,n,k,data) → (c,t)
Dec(c,t,k,n)
```

4) Qual é uma boa heurística para conseguir aleatoriedade num computador? Qual a sua desvantagem?

▼ Resposta

A aleatoriedade pode ser obtida por meio da avaliação de recursos computacionais (e.g temperatura, uso do processador). O problema desta abordagem reside no facto de que obter estes valores é um processo lento e a entropia aumenta gradualemente com o tempo, ou seja, se for requisitado a obtenção destes valores constantemente pode-se perder um certo grau de aleatoriedade.

5) Compare `random` com `urandom` . Qual dos dois é mais recomendado? Por que?

▼ Resposta

`random` e `urandom` são dois ficheiros no sistema linux capazes de gerar valores aleatórios a partir de medidas de recursos computacionais (uma heurística). O `random` bloqueia até que uma certa entropia seja adquirida. `urandom` por outro lado retorna independente da entropia. Entre os dois ficheiros o `urandom` é o mais recomendado, pois não se sabe muito bem o quão bem o computador consegue dizer se uma entropia é “boa” ou “não”. Ainda, muitas vezes nas aplicações atuais não se lida com o facto de que `random` retorna nada se uma determinada entropia não é atingida e por isso acaba-se por não realizar nenhum tipo de aleatoriedade na encriptografia. Portanto, o `random` além de se correr o risco de deixar o sistema mais lento devido a espera dos níveis de entropia serem satisfatórios, pode ocorrer da aplicação na verdade não gerar nenhuma aleatoriedade para cifrar o conteúdo. Neste sentido, mais vale ter pouca aleatoriedade do que não ter nenhuma.

6) O que é cookie poisoning? Cite um exemplo.

▼ Resposta

Cookie poisoning consiste em alterar a cookie de forma que quando o servidor requisitá-lo a informação obtida seja a que o atacante deseja.
Imagine que o atacante é um jogador de jogo. O servidor para envia cookies para os clientes para evitar com que toda a informação seja armazenada nele. Um cookie em particular armazena os pontos de um jogador. Ao alterar a sua informação nesta cookie (i.e quantos pontos o jogador tem), quando o servidor requisitar o cookie novamente a informação implantada pelo jogador será considerada e processada pelo servidor.