Sistema de Mensagens Seguras

Diogo Ferreira, 76425
Pedro Martins, 76551

Segurança 2017/2018

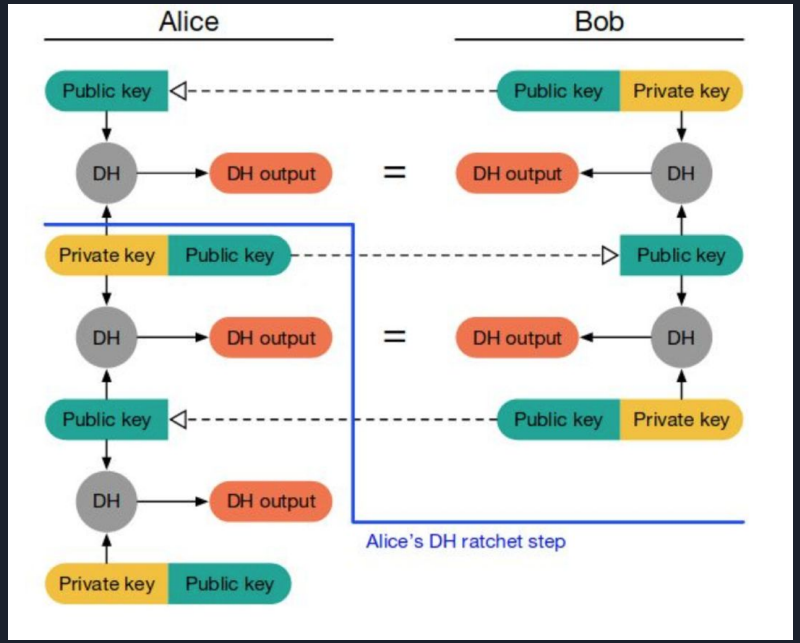
Cipher suite

- *KKK-SSS_MMM-AAA_PPP-XXX_NNN_HHH_CCC_HHH-III-HHH*
  - *KKK* - algoritmo de troca de chaves de sessão (Ratchet Diffie-Hellman);
  - *SSS_MMM* - algoritmo de cifra simétrica e o seu modo (AES192 e AES256, CBC e CTR);
  - *AAA_PPP* - algoritmo de cifra assimétrica e o seu padding (RSA1024 e RSA2048, OAEP e PKCS1v15);
  - *XXX_NNN_HHH_CCC_HHH* - algoritmo usado para assinaturas e verificação das mesmas (RSA2048, PSS-SHA256 ou SHA384 para o servidor, PKCS1v15-SHA256 para o CC);
  - *III* - algoritmo de controlo de integridade (HMAC);
  - *HHH* - algoritmo de hashing (SHA256 e SHA384);

Troca de Chaves de Sessão

- Ephemeral Elliptic Curve Diffie-Hellman Exchange:
  - Criação de um canal seguro entre servidor e clientes;
  - EECDHE permite um mesmo nível de segurança que outros sistemas criptográficos assimétricos (e.g., RSA) com chaves mais pequenas;
  - Operações com curvas elípticas são, por norma, mais rápidas;
  - Mais eficiente do que o clássico algoritmo Diffie-Hellman.
- Ratchet Diffie-Hellman:
  - Segue as mesmas ideias que EECDHE;
  - A cada mensagem trocada é gerado um novo segredo;
  - Garante *forward and backward secrecy* a cada mensagem trocada e não a cada sessão;
- Uso de um salt (combinação de cliente e servidor) para evitar ataques de repetição;
- Uso de uma Key Derivation Function (HKDF) para obter chaves com determinados tamanhos;
- Usam-se múltiplas derivações caso o cliente envie mensagens sem obter respostas do servidor.

Ratchet Diffie-Hellman



Autenticação dos Intervenientes

- Uso de assinaturas RSA (2048 bits) para autenticação dos intervenientes e controlo de integridade (apenas nas duas primeiras mensagens);
- Evita-se assim ataques MiTM;
- CC usa RSA2048 com padding PKCS1v15-SHA256;
- Servidor usa RSA2048 com padding PSS-SHA256 ou 384;
- Chaves e certificados do servidor gerados usando XCA.

Autenticação dos Intervenientes

- A mensagem de um interveniente só será validada se:
  - O seu certificado estiver presente na mensagem;
  - O certificado já estiver validado em cache ou...
  - Toda a cadeia de certificação não tenha sido revogada ou esteja expirada:
    - Verifica-se por OCSP, caso esteja disponível;
    - Caso contrário, descarregam-se as CRL e as delta mais recentes;
  - Todas as assinaturas da cadeia de certificação sejam validadas;
  - A assinatura da mensagem for válida.
  - O MAC também deve ser validado.
- Uso de HMAC para controlo de integridade e de sequencialidade e frescura das mensagens:
  - Primeira mensagem pelo cliente enviada apenas com assinatura + *nonce*;
  - Primeira mensagem enviada pelo servidor com assinatura + HMAC(message + *nonce*, session_key);
  - Seguintes mensagens com HMAC(message + last_HMAC, session_key).

Processo de Handshake

- Primeira mensagem enviada pelo cliente (init):

```
{
    "type": "init",
    "payload": base64.b64encode(json.dumps({
        "uuid": <user UUID>,
        "secdata": {
            "dhpubvalue": <diffie-hellman public value>,
            "salt": <128 bit random value>,
            "index": <number of hash derivations>
        },
        "nonce": <128 bit random value>
    }).encode()).decode(),
    "signature": <payload signed by authentication private key>,
    "certificate": <authentication public key certificate>,
    "cipher_spec": <used cipher specification>
}
```

Processo de Handshake

- Primeira resposta do servidor já é segura (cifrada e autenticada):

```
    "type": "secure",
    "payload": base64.b64encode(json.dumps({
        "message": <ciphered payload of the message>,
        "secdata": {
            "dhpubkey": <public value of diffie-hellman exchange>,
            "salt": <salt used on diffie-hellman>,
            "iv": <AES initialization value>,
            "index": <number of key derivations>
        }
    }).encode()).decode(),
    "mac": <hmac(nonce|payload, session_key)>,
    "signature": <payload signed by authentication private key>,
    "certificate": <authentication public key certificate>,
    "cipher_spec": <used cipher specification>
}
```

Processo de Handshake

- Restantes mensagens trocadas entre servidor e cliente:

```
"type": "secure",
"payload": base64.b64encode(json.dumps({
    "message": <ciphered payload of the message>,
    "secdata": {
        "dhpubkey": <public value of diffie-hellman exchange>,
        "salt": <salt used on diffie-hellman>,
        "iv": <AES initialization value>,
        "index": <number of key derivations>
    }
}).encode()).decode(),
"mac": <hmac(last_received_hmac|payload, session_key)>,
"cipher_spec": <used cipher specification>
```

Mensagens RESOURCE

- Requisição ao servidor de chaves públicas, certificados e *cipher suites* de outros clientes;
- Encapsuladas em mensagens secure ou embutidas noutro tipo de mensagens;
- Aquando a sua receção, o certificado é validado assim como a respetiva assinatura;

Pedido:

```
{
    "type": "resource",
    "ids": [<identifier of the user>, ...]
}
```

Resposta:

```
{
    "result": [
        {
            "id": <identifier of the user>,
            "secdata": base64.b64encode(json.dumps({
                "rsapubkey": <RSA generated public key>,
                "cccertificate": <CC authentication public key certificate>,
                "cipher_spec": <cipher spec choosed by the client>
            }).encode()).decode(),
            "signature": <signature over secdata by CC authentication key>
        }, ...
    ]
}
```

Criação de utilizadores

- A cada utilizador ser-lhe-á associado:
  - Um par de chaves RSA e o respetivo certificado para assinatura (chaves e certificado de Autenticação do CC);
  - Um par chaves de RSA usadas para cifra e decifra de conteúdo que lhe é destinado (e.g., chaves e IV de cifra simétrica) - a chave privada fica guardada no sistema de ficheiros protegida por password;
  - Um UUID, que resulta da geração de uma síntese do certificado da chave de autenticação do CC;
  - Um *cipher spec* que escolheu aquando do seu registo na plataforma;

```
{
    "type": "create",
    "uuid": <user uuid>,
    "secdata": base64.b64encode(json.dumps({
        "rsapubkey": <RSA generated public key>,
        "cccertificate": <CC authentication public key certificate>,
        "cipher_spec": <cipher spec choosed by the client>
    }).encode()).decode(),
    "signature": <signature over secdata by CC authentication key>
}
```

Criação de utilizadores

- O servidor, se o utilizador ainda não existir no sistema, e depois de validar o certificado do cliente, irá registar um novo perfil de utilizador com os seguintes campos :

```
{
    "id": <identifier>,
    "uuid": <user universal identifier>,
    "mbox": <message box path>,
    "rbox": <sent box path>,
    "secdata": base64.b64encode(json.dumps({
        "rsapubkey": <RSA generated public key>,
        "cccertificate": <CC authentication public key certificate>,
        "cipher_spec": <cipher spec choosed by the client>
    }).encode()).decode(),
    "signature": <signature over secdata by CC authentication key>
}
```

Controlo de acesso

- Em todas as mensagens recebidas pelo servidor, o mesmo valida o ID do cliente da socket;
- Nas mensagens NEW e ALL, verifica se o ID é igual ao ID da *message box* à qual o cliente pretende aceder, ou no caso de ser uma mensagem STATUS, a respetiva *receipt box* (ou seja, o do campo “id” da mensagem);
- Nas mensagens RECV e RECEIPT, é verificado o campo “id” e também impedido a um cliente com ID diferente de receber ou enviar um recibo em nome de outrém;
- Nas mensagens SEND, é verificado o campo “src” e evita-se assim enviar mensagens impersonando outros clientes.

Troca de Mensagens entre Utilizadores

- Uso de mensagens RESOURCE para obter informações de outros utilizadores;
- Uso do *cipher suite* do destinatário para cifrar a mensagem;
- Gerar um *nonce* (valor aleatório de 128bits) - usado para validação de recibos (evita que uma mensagem com o mesmo conteúdo tenha criptogramas iguais);
- Gerar uma chave de cifra simétrica e um IV;
- Cifrar a mensagem;
- Cifrar a chave|IV|nonce com a chave pública RSA do destinatário;
- Assinar o payload com a chave de autenticação do CC do remetente.

```
"payload": {
    "src": <id of the user that sent the message>,
    "dst": <id of the destination user>,
    "message": <ciphered message>,
    "nonce_key_iv": <ciphered iv|key|nonce used to cipher original message>
},
"signature": <payload signed by authentication private key>,
"cipher_spec": <used cipher specification>
```

Troca de Mensagens entre Utilizadores

- O certificado não vai na mensagem SEND, nem ficará armazenado com a restante mensagem, porque o servidor já o tem na descrição do cliente;
- Quando um cliente envia um RECV, o servidor carrega o ficheiro da mensagem e envia também o certificado do cliente.

```
"payload": {
    "src": <id of the user that sent the message>,
    "dst": <id of the destination user>,
    "message": <ciphered message>,
    "nonce_key_iv": <ciphered iv|key|nonce used to cipher original message>
},
"signature": <payload signed by authentication private key>,
"cipher_spec": <used cipher specification>
```

Troca de Mensagens entre Utilizadores

- O processo de cifra da mensagem é executado duas vezes, uma para guardar na *message box* e outra na *receipt box*;
- Apenas o *nonce* é gerado uma vez e é igual nas duas mensagens;
- Quando um cliente envia um RECV, o servidor carrega o ficheiro da mensagem e envia também todas as informações do remetente (mensagem RESOURCE embutida);
- Na receção, é feito o processo inverso do envio, além de que se tem de verificar a validade do certificado do remetente e da respetiva assinatura da mensagem.

Geração de recibos

- Após a decifra de uma mensagem, o *nonce* é guardado para ser enviado no recibo de leitura;
- É criado um dicionário JSON com o ID do emissor e destinatário, a mensagem decifrada, o *timestamp* da criação do recibo e o *nonce* retirado da mensagem original;
- O valor anterior é assinado pela chave de autenticação do CC;

```
{
    "src": <id of the user that sent the message>,
    "dst": <id of the destination user>,
    "message": <deciphered message>,
    "timestamp" <timestamp of the moment of receipt generation>,
    "nonce": <nonce grabbed from read message>
}
```

- É criado um novo dicionário, resultante da junção do *timestamp* e da assinatura sobre dicionário anterior usando a chave de autenticação do CC do destinatário da mensagem;

```
{
    "timestamp" <timestamp of the moment of receipt generation>,
    "signature": <signature over receipt contents by CC authentication key>
}
```

Geração de Recibos

- Tal como qualquer outra mensagem destinada a um cliente, este dicionário é cifrado para ser guardado de maneira segura na *receipt box* do emissor da mensagem.
- Cifrar a chave|IV com a chave pública RSA do destinatário do recibo;

```
{
    "receipt": <ciphered receipt>,
    "key_iv": <ciphered key|iv used to cipher receipt>,
    "cipher_spec": <used cipher specification>
}
```

Verificação do Estado de Mensagens / Recibos

- Decifrar a mensagem presente na *receipt box* e obter o nonce original;
- Decifrar o recibo;
- Validar o certificado do emissor do recibo (campo “dst”), se este não estiver em cache;
- Reconstruir o dicionário original JSON:

```
{
    "src": <id of the user that sent the message>,
    "dst": <id of the destination user>,
    "message": <deciphered message>,
    "timestamp" <timestamp of the moment of receipt generation>,
    "nonce": <nonce grabbed from read message>
}
```

- Validar a assinatura de cada recibo cifrado;
- Caso a mesma seja válida, apresentar o recibo como válido, assim como o respetivo *timestamp* e assinatura.