Universidade de Aveiro

departamento de electrónica, telecomunicações e informática

Segurança

Sistema de Mensagens Seguras

Milestone 1

8240 - mestrado integrado em engenharia de computadores e telemática

Diogo Ferreira
NMec: 76425 | P1G1

Pedro Martins
NMec: 76551 | P1G1

2017-2018

Conteúdos

Capítulo

Introdução

Nos dias de hoje, a segurança nas redes informáticas é um tema controverso, sendo necessária uma correta definição e implementação das suas políticas.

Neste trabalho, pretende-se desenvolver um sistema de mensagens seguras entre diversos utilizadores, autenticados com o seu Cartão de Cidadão. Vários mecanismos foram implementados, tais como autenticação dos intervenientes através de assinaturas digitais, derivação de chaves de sessão entre clientes e servidor para criação de um túnel seguro, entre outras, que serão descritas neste relatório.

Capítulo

Especificações da Plataforma

1 Algoritmos/Sistemas de Cifra

Na troca de informação entre servidor e clientes, é sempre definido o *"cipher suite"* que será usado. Ele seguirá o formato *XXXX-SSS_MMM-AAA_PPP-HHH*, correspondente a:

- **XXXXX**, o algoritmo de troca de chaves entre servidor e cliente:
  - Ephemeral Eliptic Curve Diffie-Hellman (EECDH). O uso de curvas elípticas permite um mesmo nível de segurança com chaves mais pequenas que outros sistemas criptográficos assimétricos, como Rivest–Shamir–Adleman (RSA). Além disso, operações com curvas elípticas são, por norma, mais rápidas; Eliptic Curve Diffie-Hellman (ECDH) é mais rápido do que Diffie-Helman. Por fim, o uso de um valor aleatório e a constante mudança de chaves permite evitar ataques de repetição. Em conjunção com o RSA, para autenticação dos intervenientes, podemos chamar a este processo de EECDH.
- **SSS_MMM**, o algoritmo de cifra simétrica e o seu modo, usado para cifrar mensagens. Optou-se apenas por usar AES192 e AES256 porque, apesar de uma maior perda de performance em relação a AES128, o aumento de segurança é compensatório. Optou-se ainda pelo uso de modos de cifra contínuas, evitando-se a manutenção de padrões da mensagem original:
  - AES_CBC, que não reproduz padrões, tem confusão na entrada da cifra, a alteração determinística de bits é complicada, permite paralelização na decifra e recuperação após perdas de blocos inteiros;

  - AES_CTR, que não reproduz padrões e tem confusão na entrada da cifra (considerando o Initialization Vector (IV) secreto) e permite paralelização na cifra e na decifra.

- **AAA_PPP**, o algoritmo de cifra assimétrica e o seu modo de *padding*, usado para cifrar chaves (chaves híbridas). Optou-se apenas pelo uso de RSA (1024 e 2048 bits) face ao ElGamal, uma vez que este apenas usa uma complexidade relacionada com logaritmos modulares, enquanto que o primeiro usa também fatorização. Em relação aos modos de *padding*:
  - RSA2048_OAEP, o recomendado para sistemas atuais;
  - RSA1024_PCKS1v15, não recomendado para sistemas atuais, mas que pode atuar para retro compatibilidade.

- **HHH**, o algoritmo de *hashing* usado para síntese de mensagens para usado em assinaturas, por exemplo. Usaram-se funções de síntese de vários tamanhos, para quando possível usar um valor maior, para diminuir a probabilidade de colisões. Portanto:
  - SHA256;
  - SHA384;

Quanto aos algoritmos de assinatura digital usaremos o RSA (2048 bits), principalmente porque este ser o sistema usado pelo Cartão de Cidadão (CC).

Resumindo, a especificações de cifra suportadas são:

- EECDH-AES192_CFB-RSA1024_PCKS1v15-SHA256;
- EECDH-AES192_CFB-RSA2048_OAEP-SHA256;
- EECDH-AES256_CFB-RSA2048_OAEP-SHA384;
- EECDH-AES192_CTR-RSA1024_PCKS1v15-SHA256;
- EECDH-AES192_CTR-RSA2048_OAEP-SHA256;
- EECDH-AES256_CTR-RSA2048_OAEP-SHA384;

2 Processo de *Handshake* entre Cliente-Servidor

Para que as trocas de informação entre cliente e servidor possam ser feitas de forma confidencial e autenticada, é primeiramente necessário realizar um processo de *handshake* entre ambos, garantindo assim um canal seguro de comunicação.

Para tal, é necessária uma troca de chaves para que, no final deste processo, os dois intervenientes tenham um segredo comum partilhado com o qual irão cifrar todas as mensagens, evitando assim *eavesdroping attacks* e observadores passivos. No entanto, ataques Man-in-The-Middle (MiTM) não serão prevenidos, os quais apenas serão contornados com a autenticação entre as partes, usando chaves privadas e públicas, assim como os respetivos certificados dos intervenientes.

Todo este processo de autenticação e troca de chaves é designado por Ephemeral Eliptic Curve Diffie-Hellman Exchange (EECDHE).

2.1 Autenticação das Partes

Quando um cliente se tenta ligar ao servidor, o primeiro passo será enviar o *cipher spec* suportado. O servidor ao receber esta mensagem, compara o *cipher spec* requisitado e, caso o suporte, confirma a sua utilização. Caso o *cipher spec* enviado pelo cliente não seja suportado, o servidor envia uma proposta para cliente e aguarda pela sua resposta.

Uma ligação segura, por si só, não garante autenticação entre intervenientes, nem previne um ataque MiTM. Para tal, é necessária a troca de certificados de autenticação e proceder à verificação das chaves públicas e assinaturas das partes.

O CC usa chaves assimétricas RSA de 2048 bits, portanto, assumir-se-á que o servidor também funcionará de maneira equivalente, sendo a chave e os certificados do mesmo gerados usando X Certificate and key management (XCA).

O primeiro passo é enviar os certificados e verificar a validade dos mesmos. Usando Online Certificate Status Protocol (OCSP) e o *link* presente no certificado, é verificada a sua validade. Caso seja válido, o processo continua, senão este é abortado. O formato das mensagens trocadas será o seguinte:

```
{
    "type": "insecure",
    "certificate": <public key authentication certificate>
}
```

2.2 Troca de Chaves de Sessão

Para cada sessão, o servidor e o cliente têm de estabelecer uma chave de sessão para estabelecer uma ligação segura. O processo de *handshake* seguirá as linhas do EECDHE, como referido anteriormente.

O processo consiste numa troca de chaves públicas sobre um canal inseguro de modo a que, no final do processo, ambos possuam um segredo comum, do qual se poderá derivar uma chave para uso em sistemas criptográficos simétricos (como Advanced Encription System (AES)) para troca de mensagens. Como o segredo comum pode não ser, em princípio, resistente a ataques *brute force*, é necessário usar-se uma Key Derivation Function (KDF) (um processo também conhecido como *key stretching*, de tal forma a obter uma chave de determinado tamanho a partir de outra para ser usada em outros algoritmos criptográficos). No contexto deste trabalho, usar-se-á Hash-based Message Authentication Key Derivation Function (HKDF), pois permite-nos, de forma facilitada, evitar os ataques de repetição, através do uso de um *salt* (valor aleatório) e derivar a chave para usar na cifra das mensagens.

Mais especificamente, o processo utilizado é conhecido como Ratchet Diffie-Hellman Exchange (RDHE), no qual se deriva um novo segredo a cada mensagem trocada e não a cada sessão.

Como a primeira mensagem de cada interação é sempre enviada pelo cliente, o servidor gera apenas um valor público inicialmente, logo:

1. O servidor, depois de chegar a acordo com o cliente sobre o *cipher spec* a ser usado, gera um valor público e privado para troca das mesmas via EECDH e um valor aleatório (*salt*);

2. O cliente, ao de receber a confirmação do *cipher spec* a usar, gera também uma chave pública e privada (ECDH) e um valor aleatório para evitar ataques de repetição (*salt*);

3. Tendo o valor público do servidor e o seu valor privado, o cliente pode gerar um segredo comum, que advém da combinação da sua chave privada e da chave pública do servidor;

4. A partir de uma combinação dos valores de *salt* (evita ataques de repetição) de ambos e do segredo comum, o cliente deriva uma chave de cifra para uso no envio da sua próxima mensagem;

5. Juntamente com a sua mensagem, o cliente envia para o servidor o seu valor público e o seu *salt*;

6. O servidor, ao receber a mensagem do cliente, deriva a chave usada para cifrar a mensagem, através do seu valor privado e do novo valor privado recebido do cliente, assim como ambos os *salt*;

7. Para enviar a resposta ao cliente, o servidor gera um novo par de valores, deriva um novo segredo comum, resultante da combinação do seu novo valor privado, do valor público recebido do cliente, do seu novo *salt* e do *salt* recebido do cliente;

8. Tal como o cliente, o servidor envia juntamente com a mensagem o seu novo valor público e o seu novo valor aleatório;

Este processo repete-se enquanto houver mensagens trocadas entre um cliente e o servidor. Este processo é descrito na Figura .1:

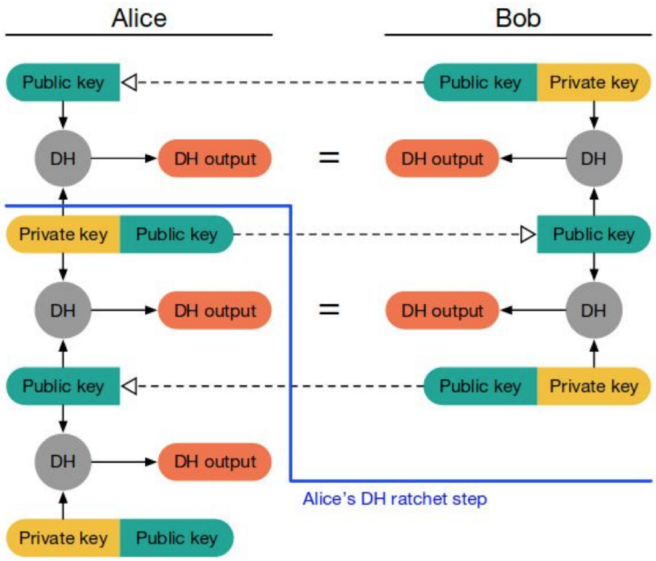

Figura .1: Processo de troca de chaves RDHE.

A primeira mensagem trocada, enviada do servidor para o cliente, segue o seguinte formato, aproveitando o facto de inicialmente o cliente já ter enviado o seu certificado e, ao enviar o seu, envia já os valores para a troca de chaves de sessão:

```
{
    "type": "insecure",
    "pubvalue": <diffie-hellman public value>,
    "salt": <random value>,
    "certificate": <public key authentication certificate>
}
```

Depois deste processo, e existindo um segredo comum (chave para uso em sistemas criptográficos simétricos) partilhado por ambos, pode-se confirmar que está estabelecida uma ligação segura e as mensagens entre cliente e servidor podem ser cifradas segundo o *cipher spec* acordado. A partir desse momento, *eavesdroping attacks* e observadores passivos já não serão um problema.

O RDHE garante *forward and backward secrecy*, visto que as chaves usadas na troca de uma mensagem é independente das chaves usadas nas mensagens seguintes ou anteriores, logo, no caso de ser descoberta uma chave, não será possível obter informações relativas a outras trocas de informação.

Para finalizar, todas as mensagens trocadas entre cliente e servidor são concatenadas com o *timestamp* do momento de envio e passadas a uma função de síntese, sendo o resultado da mesma anexado ao pacote da mensagem e guardado numa lista no próprio cliente. Quando o servidor envia a resposta a um pedido do cliente, deve anexar o *hash* que o cliente lhe enviou para que, quando o utente receba a mensagem, possa verificar se a mesma é a resposta a um pedido que consta na sua lista de mensagens enviadas; ao receber uma resposta a um determinado pedido, a síntese correspondente é retirada da lista.

Logo, a estrutura do pacote depois de estabelecida a sessão segura será:

```
{
    "type": "secure",
    "payload": {
        "message": <ciphered payload of the message>,
        "hash": <digest of (message|timestamp) to match response>,
        "secdata": {
            "dhpubkey": <public value of diffie-hellman exchange>,
            "salt": <salt used on diffie-hellman>
        }
    },
    "signature": <payload signed by authentication private key>,
    "certificate": <authentication public key certificate>
}
```

3 Criação de Utilizadores

No contexto da aplicação, para que um utilizador possa enviar e receber mensagens, é necessário primeiro registar-se no servidor para que lhe sejam atribuídas uma *message box* e uma *receipt box*.

Antes de comunicar com o servidor, deve ser gerado um par de chaves para cifrar as mensagens que serão armazenadas no servidor, sendo portanto o primeiro passo a geração de um par de chaves assimétricas RSA, que serão usadas para cifrar e decifrar as chaves AES e o IV, que servirão para cifrar as mensagens dos utilizadores e os *nounces* usados para validar o envio de recibos.

As chaves privadas anteriormente geradas necessitam de ser guardadas de forma segura, logo cada utilizador terá de usar um *PIN* ou *password* para derivar a chave que irá cifrar e decifrar o valor privado. Para o efeito, usar-se-á HKDF que, a partir da *password* e de um *salt* gerado aleatoriamente, deriva uma chave para cifrar a chave privada RSA. Por fim, é criado um ficheiro onde é armazenada a chave cifrada, concatenada com o respetivo *salt*.

De uma maneira sumária, cada utilizador terá:

- Um par de chaves RSA para cifra e decifra de mensagens e *nounces*, usados na validação de recibos;
- Um par de chaves RSA do respetivo CC, usadas para assinar conteúdo.

Para a efetiva criação de um utilizador no sistema, é necessário enviar uma mensagem de *payload* `CREATE` para o servidor com:

- O UID do utilizador, que consiste numa síntese do certificado da chave pública, extraído do CC;
- Um campo (`secdata`) que engloba o certificado da chave pública de autenticação do CC e a chave pública RSA gerada pelo cliente.

O formato da mensagem será:

```
{
    "type": "create",
    "uuid": <user uuid>,
    "secdata": {
        "rsapubkey": <RSA generated public key>,
        "cccertificate": <CC authentication public key certificate>
    }
}
```

O servidor, ao processar a mensagem `CREATE`, irá, primeiramente, verificar se o `uuid` já existe na sua base de dados. Se tal não se verificar, irá passar à validação do certificado usando OCSP. Caso esta validade se verifique, 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": {
        "rsapubkey": <RSA generated public key>,
        "cccertificate": <CC authentication public key certificate>
    }
}
```

4 Mensagens RESOURCE

No contexto de comunicação segura entre clientes, foi necessário introduzir um novo tipo de mensagens trocadas entre cliente e servidor, RESOURCE, para que estes possam pedir recursos ao servidor que necessitam para gerar a mensagem que realmente pretendem enviar.

Estas têm como intuito eventuais pedidos de chaves públicas e certificados que o cliente faça ao servidor, pois este último armazena chaves públicas RSA e certificados do CC dos utentes, e o cliente que faz o pedido necessita dos mesmas para cifras, decifras, validação de assinaturas, entre outros.

O cliente também terá ao dispor uma *cache* para que possa armazenar temporariamente chaves e certificados de outros utentes, evitando assim ter de realizar este pedido ao servidor a cada troca de mensagens que se dê.

O formato de uma mensagem RESOURCE será:

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

O servidor irá responder com:

```
{
    "result": [
        {
            "id": <identifier of the user>,
            "rsapubkey": <RSA generated public key>,
            "cccertificate": <CC authentication public key
                certificate>
        }, ...
    ]
}
```

5 Envio de Mensagens para Outros Utilizadores

Quando se pretende enviar uma mensagem para um utilizador, é primeiramente necessário cifrar a respetiva mensagem, para que apenas o destinatário e o emissor na sua a possam ler. No entanto, as chaves utilizadas para cifrar os conteúdos terão de ser diferentes, isto é, o emissor apenas poderá decifrar a mensagem da sua *receipt box* e não a presente na *message box* do destinatário, e vice-versa. Para tal, apenas o destinatário deverá ter acesso ao *nounce* guardado na mensagem, pois apenas este pode enviar recibos.

O processo de envio de uma mensagem segue as seguintes instruções:

1. Gerar a síntese da mensagem e, concatenando o resultado da mesma com o *timestamp* do momento de envio, gerar nova síntese. O resultado desta última é cifrado com a chave pública RSA do destinatário, resultando num *nounce*;
2. Gerar uma chave de cifra simétrica e um IV, segundo o algoritmo *cipher spec* escolhido pelo emissor;
3. Cifrar a mensagem (cifra simétrica) usando a chave e o IV gerados;
4. Cifrar a chave e o IV (concatenados com um separador) com a chave pública RSA presente no perfil do destinatário;
5. Cifrar o *nounce* da mesma maneira;
6. Concatenar os resultados anteriores com a mensagem cifrada e o *cipher spec* usado, separados por um separador específico (eg., dois *carriage returns*).
7. Por fim, a mensagem é assinada com a chave privada de autenticação do CC e concatenada à mensagem.

Considerando para o efeito o separador `\n\t`, o resultado da mensagem seria:

```
nounce \n\t cipher_spec \n\t key_iv \n\t ciphered_message \n\t
    signature
```

No entanto, como é necessário guardar também a mensagem na caixa de recibos do emissor, todos os passos anteriormente descritos devem ser repetidos, excetuando a geração do *nounce*, mas, desta vez, cifrando-o com a chave pública RSA do próprio remetente.

Por fim, as duas "mensagens"são concatenadas com um separador específico, encapsuladas num pacote `secure` (como anteriormente referido), e enviadas para o servidor. O servidor ao receber a mensagem, separa-as e guarda-as na *message box* do destinatário e na *receipt box* do emissor, respetivamente.

6 Leitura de Mensagens

Quando um utilizador pretende ler uma das mensagens na sua *message box*, é necessário:

1. Escolher a mensagem a ler e enviar o respetivo pedido ao servidor;
2. O servidor irá carregar o ficheiro da *message box* do utilizador e o certificado do CC presente no perfil do remetente e enviar ambos para o cliente;

O cliente ao receber a mensagem:

1. Valida o certificado usando OCSP;
2. Separa a assinatura da restante mensagem e, juntamente com o certificado recebido, verifica a sua validade;
3. Obtém os restantes componentes: o *cipher spec*, as informações da cifra e a mensagem cifrada;
4. Separa o *nounce* (guardado para posterior envio de um recibo), a chave AES e o IV, depois de decifrados com a sua chave privada RSA (usando a sua *password* e carregando a chave cifrada armazenada em ficheiro);
5. Em concordância com o *cipher spec* da mensagem, decifra a mensagem com a chave e IV obtidos anteriormente.

7 Envio e Receção de Recibos

Depois da leitura de uma mensagem, pode ser produzido um recibo de leitura da mesma, mas, para tal, é necessário o *nounce* com o qual a mesma vem concatenada.

Para enviar um recibo, é gerada a síntese da mensagem e, concatenando o resultado da mesma com o *timestamp* do momento de envio, gerar nova síntese. O resultado desta última é assinada pela chave privada de autenticação do seu CC. Por fim, é gerada uma assinatura da mesma a partir da chave privada de autenticação do seu CC.

Considerando para o efeito o separador `\n\t`, o resultado da mensagem seria:

```
hashed_timestamp_message \n\t signature
```

Por fim, tal como uma mensagem, também o recibo deve ser cifrado para que apenas o emissor da mensagem original tenha acesso a ele, sendo necessário:

1. Gerar uma chave de cifra simétrica e um IV, segundo o algoritmo *cipher spec* escolhido anteriormente pelo emissor;
2. Cifrar a mensagem (cifra simétrica) usando a chave e o IV gerados;
3. Cifrar a chave e o IV(concatenados com um separador) com a chave pública RSA presente no perfil do remetente;
4. Cifrar o *nounce* obtido da leitura da mensagem da mesma maneira;
5. Concatenar os resultados anteriores com a mensagem cifrada, o *cipher spec* usado, separados por um separador específico (eg., dois *carriage returns*).
6. Por fim, toda a mensagem é assinada com a chave privada de autenticação do CC e concatenada à mensagem.

A estrutura do recibo cifrado seria:

```
nounce \n\t cipher_spec \n\t key_iv \n\t ciphered_receipt \n\t
    signature
```

Quando o servidor recebe um recibo de um leitor, deve:

1. Validar o certificado usando OCSP;
2. Separar a assinatura da restante mensagem e, juntamente com o certificado recebido, verifica a sua validade;

3. Obter o *nounce* e compará-lo com o guardado na mensagem correspondente na *receipt box* do remetente;

4. Se a validade do *nounce* se verificar, o recibo é guardado na *receipt box* do remetente.

8 Verificação do Estado de Recibos

Quando um utilizador pretende verificar o estado de uma mensagem na sua *receipt box*, basta enviar um pedido com o `id` da caixa e o `msg_id`.

O servidor ao receber o pedido, carrega o ficheiro da mensagem da *receipt box* do utilizador e todos os recibos e envia-os para o cliente.

O cliente deve decifrar a mensagem e os recibos. O processo de decifra da mensagem e dos recibos segue exatamente o mesmo processo referido anteriormente.

Capítulo

Conclusão

Neste relatório pretendeu-se elaborar um plano para a elaboração de um sistema de comunicação de mensagens seguras entre utilizadores, no entanto, o plano e o sistema poderão sofrer alterações, se assim for necessário.

Acrónimos

MiTM Man-in-The-Middle

RSA Rivest–Shamir–Adleman

ECDH Eliptic Curve Diffie-Hellman

EECDH Ephemeral Eliptic Curve Diffie-Hellman

EECDHE Ephemeral Eliptic Curve Diffie-Hellman Exchange

RDHE Ratchet Diffie-Hellman Exchange

AES Advanced Encription System

KDF Key Derivation Function

HKDF Hash-based Message Authentication Key Derivation Function

IV Initialization Vector

CC Cartão de Cidadão

OCSP Online Certificate Status Protocol

XCA X Certificate and key management

Bibliografia

[1] André Zúquete and João P. Barraca. Slides segurança 2017-2018. [Online; acedido em Novembro 2017].

[2] Individual Contributors. Cryptography.io. [Online; acedido em Novembro 2017].

[3] Robert Picciotti. Signal & the double ratchet. [Online; acedido em Novembro 2017].

[4] Paul Bakker. Why use ephemeral diffie-hellman. [Online; acedido em Novembro 2017].

[5] Entrust. Zero to ecdh in 30 minutes. [Online; acedido em Novembro 2017].