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 | P1G12

Pedro Martins
NMec: 76551 | P1G12

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. Para controlo de integridade das mesmas, todas as mensagens cifradas com Advanced Encription System (AES), garantindo assim autenticação e controlo de integridade:
  - 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 (processo de cifra híbrido). 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 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 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.

O processo é descrito a seguir:

1. O cliente gera um par de valores ECDH e um valor aleatório (*salt*) e envia o seu valor público e o *salt* para o servidor; a primeira mensagem é a única que não terá o seu conteúdo cifrado;

2. O servidor, ao receber a mensagem, gera também uma chave pública e privada (ECDH) e um valor aleatório para evitar ataques de repetição (*salt*);

3. O servidor irá também gerar um valor aleatório (`client_id`) para identificar o cliente e irá criar uma entrada num dicionário onde armazenará os valores ECDH para o referido cliente;

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

5. A partir de uma combinação dos valores de *salt* (evita ataques de repetição) de ambos e do segredo comum, o servidor deriva uma chave de cifra para uso no envio da sua próxima mensagem; a partir deste momento a comunicação será cifrada (mas não autenticada);

6. Juntamente com a sua mensagem, o servidor envia para o cliente o seu valor público, o seu *salt* e o `client_id` correspondente;

7. O cliente, ao receber a mensagem do servidor, deriva a chave usada para cifrar a mensagem, através do seu valor privado e do novo valor público recebido do servidor, assim como de ambos os *salt*; a geração desse segredo leva à criação de um *index* que será incrementado a cada geração de segredos comuns e que não será reinicializado até receber um novo valor público do servidor (isto garante que o processo de troca de chaves é sempre coerente entre as partes, mesmo que cliente ou servidor enviem múltiplas mensagens sem obter resposta); esse *index* irá indicar o número de vezes que uma chave é derivada a partir dos mesmos valores ECDH e deverá ser enviado para o servidor para o mesmo efetuar o mesmo número de derivações;

8. Na próxima mensagem enviada pelo cliente, é gerado um novo par de valores e um novo *salt*, e é usado o novo valor privado com o público recebido do servidor para gerar o segredo comum; o cliente tem também de enviar o seu `client_id` para o servidor identificar com que cliente está a trocar valores ECDH.

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

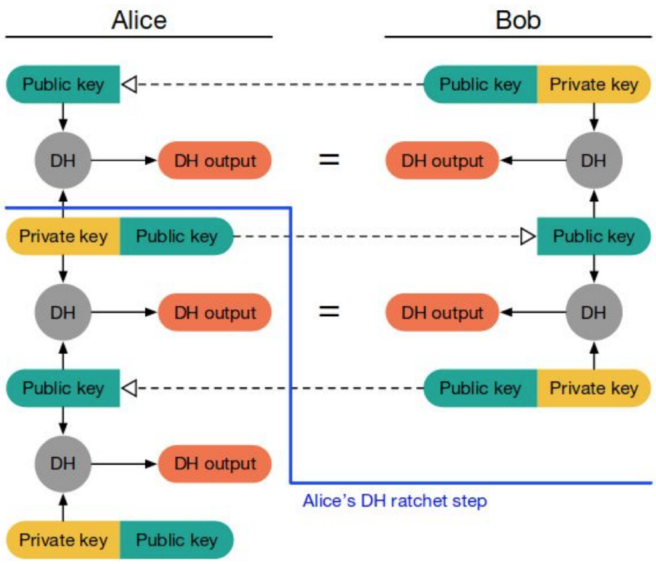

Figura .1: Processo de troca de chaves RDHE.

A primeira mensagem trocada, enviada do cliente para o servidor de maneira insegura, segue o seguinte formato:

```
{
    "type": "insecure",
    "secdata": {
        "pubvalue": <diffie-hellman public value>,
        "salt": <random value>,
        "client_id": <random value for client identification>
    },
    "nounce": <128 bit random value>,
    "cipher_spec": <used cipher specification>
}
```

Depois do envio desta primeira mensagem, e existindo um segredo comum (chave para uso em sistemas criptográficos simétricos) partilhado por ambos, pode confirmar-se 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 são independentes 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. De referenciar que a derivação de uma nova chave a cada troca de mensagens é uma funcionalidade de extra, mas que impõe mais uma camada de segurança na troca de mensagens entre cliente e servidor.[1]

Para finalizar, para todas as mensagens trocadas entre cliente e servidor é gerado um *nounce* (valor aleatório de 128 bits, isto é, um valor grande para que sejam evitadas colisões, ao qual é concatenado a síntese da mensagem - nas mensagens do tipo `INSECURE` a síntese é produzida a partir de outro valor aleatório, enquanto que nas mensagens `SECURE` é produzia a partir da síntese do campo `message`), sendo o mesmo 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 *nounce* 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, o *nounce* correspondente é retirado da lista.

2.2 Autenticação das Partes

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).

Depois de um cliente enviar a primeira mensagem para o servidor (envio dos valores públicos ECDH), na primeira resposta, o servidor já irá enviar uma mensagem cifrada e assinada pela sua chave pública de autenticação, juntamente com o seu certificado. Na receção de qualquer mensagem, é primeiramente verificado se o certificado ainda não expirou e qual o seu propósito, e, de seguida, a validade é verificada junto de uma Certification Authority (CA), usando Online Certificate Status Protocol (OCSP). Caso não exista indicação de um endereço de um serviço OCSP, deve ser descarregada a Certificate Revogation List (CRL) referenciada pelo certificado. Caso seja válido, o processo continua, senão este é abortado. Logo, a estrutura do pacote depois de estabelecida a sessão segura e autenticada (i.e., a partir do envio da primeira resposta do servidor) será:

```
{
    "type": "secure",
    "payload": {
        "message": <ciphered payload of the message>,
        "nounce": <nounce obtained from 128bit_random|message hash>,
        "secdata": {
            "dhpubkey": <public value of diffie-hellman exchange>,
            "salt": <salt used on diffie-hellman>,
```

```
            "index": <number of key derivations>,
            "client_id": <random value for client identification>
        }
    },
    "signature": <payload signed by authentication private key>,
    "certificate": <authentication public key certificate>,
    "cipher_spec": <used cipher specification>
}
```

A autenticação agora referida apenas serve para autenticar clientes ao servidor e vice-versa. Caso sejam trocadas mensagens entre clientes, as mesmas também deverão ser assinadas e autenticadas entre eles para controlo de integridade e garantia de identificação da origem.

3 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>
        }, ...
    ]
}
```

De referir que todas as mensagens do tipo `RESOURCE` serão, como todas as outras, encapsuladas no `payload` das mensagens do tipo `SECURE`, referenciadas anteriormente.

4 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 efetuando os passos já referidos na secção 2.2. 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>
    }
}
```

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 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. Por outro lado, 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 um valor aleatório de 128 bits, concatená-lo à mensagem original e gerar uma síntese (especificado no *cipher suite*), resultando num *nounce*;
2. Gerar uma chave de cifra simétrica e um IV, segundo o algoritmo especificado no *cipher suite* 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. Construir uma mensagem JavaScript Object Notation (JSON) com os resultados anteriores, a mensagem cifrada e o *cipher suite*;
7. Por fim, o `payload` é assinado com a chave privada de autenticação do CC e concatenada à mensagem.

O formato resultante da mensagem seria:

```
{
    "payload": {
        "message": <ciphered message>,
        "nounce": <ciphered nounce, obtained from 128bit_random|
            message hash>,
        "key_iv": <ciphered key|iv used to cipher original message>
    },
    "signature": <payload signed by authentication private key>,
    "cipher_spec": <used cipher specification>
}
```

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*, pois, desta vez, basta cifrá-lo com a chave pública RSA do próprio remetente.

Por fim, as duas "mensagens"são englobadas noutra mensagem JSON, 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.

O formato das mensagens que será o valor do campo `message` nas mensagens do tipo `SECURE` será:

```
{
    "message_box": <message to be saved on receiver message box>,
    "receipt_box": <message to be saved on sender receipt box>
}
```

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, como já especificado em 2.2;
2. Separa a assinatura da restante mensagem e, juntamente com o certificado recebido anteriormente, verifica a sua validade;
3. Obtém os restantes componentes: o *cipher spec*, as informações da cifra, o *nounce* 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.

O formato resultante do recibo seria:

```
{
    "hashed_timestamp_message": <hash of message|timestamp>,
    "signature": <payload signed by authentication private key>
}
```

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. Criar uma mensagem JSON com os resultados anteriores, o recibo cifrado e o *cipher spec* usado;
6. Por fim, toda a mensagem é assinada com a chave privada de autenticação do CC e concatenada à mensagem.

O formato resultante do recibo cifrado seria:

```
{
    "payload": {
        "receipt": <ciphered receipt>,
        "nounce": <ciphered nounce, obtained from 128bit_random|
            message hash>,
        "key_iv": <ciphered key|iv used to cipher original message>
    },
    "signature": <payload signed by authentication private key>,
    "cipher_spec": <used cipher specification>
}
```

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

1. Validar o certificado, como descrito em 2.2;
2. Separar a assinatura da restante mensagem e, juntamente com o certificado recebido, verificar 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

CRL Certificate Revogation List

OCSP Online Certificate Status Protocol

CA Certification Authority

XCA X Certificate and key management

JSON JavaScript Object Notation

Bibliografia

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

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

[3] Individual Contributors. Cryptography.io. [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].