ECDSA (Elliptic Curve Digital Signature Algorithm)

Márcio Aurélio Ribeiro Moreira

Especialização em Segurança da Informação – Julho de 2006
Uniminas – União Educacional Minas Gerais – Uberlândia, MG – Brasil
marcio@acc.com.br

Abstract. *In this article we presented a little introduction to the elliptic curves and it use in the cryptography. We described the concepts of digital signature, we presented the algorithm ECDSA (Elliptic Curves Digital Signature Algorithm) and we make a parallel of this with DSA (Digital Signature Algorithm). Follow we presented an application developed with the purpose of using ECDSA. Finally we presented our conclusions about this algorithm.*

Resumo. *Neste artigo apresentamos uma breve introdução às curvas elípticas e sua utilização na criptografia. Descrevemos os conceitos de assinatura digital, apresentamos o algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm) e fazemos um paralelo deste com o DSA (Digital Signature Algorithm). Em seguida apresentamos uma aplicação desenvolvida com o propósito de utilizar o ECDSA. Finalmente apresentamos nossas conclusões sobre este algoritmo.*

1. Introdução

Neste artigo apresentamos o método ECDSA (Elliptic Curve Digital Signature Algorithm), um método de assinatura digital de documentos utilizando criptografia baseada em curvas elípticas.

Começamos descrevendo os fundamentos teóricos do método, destacando o funcionamento, as vantagens (pontos fortes) e os possíveis ataques que podem ser feitos a este algoritmo. Em seguida passamos a descrever uma aplicação que utiliza o método, destacando a implementação em Java da aplicação. Finalizamos o trabalho com nossa conclusão sobre o tema e as referências utilizadas e consultadas.

2. ECDSA (Elliptic Curve Digital Signature Algorithm)

A assinatura digital de um documento deve prover as características básicas das assinaturas, que são:

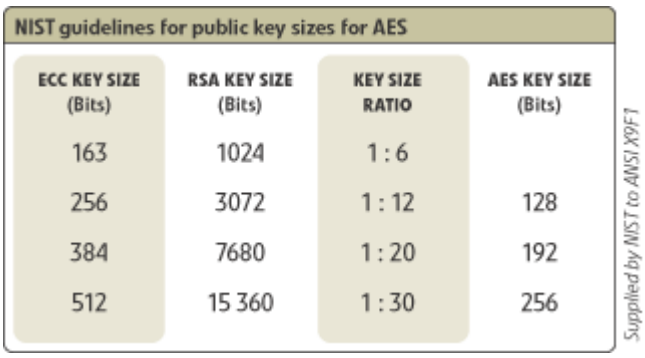
- ***Autenticidade***: a assinatura digital deve ser capaz de confirmar a autenticidade de um documento, ou seja, somente o signatário deve ser capaz de gerar sua assinatura digital para aquele documento.
- ***Não repúdio***: para um documento assinado digitalmente o autor não pode negar a autoria do mesmo.

- ***Integridade***: para sustentar as características anteriores é fundamental que um documento assinado digitalmente não possa ser adulterado sem inviabilizar o reconhecimento da assinatura digital.

Os algoritmos utilizados para assinatura digital podem ser classificados em dois grupos:

- ***Diretos***: o processo de assinatura digital envolve somente as partes envolvidas com a assinatura, a origem e o destino. Estes métodos normalmente utilizam algoritmos de criptografia simétricos. Assim, em caso de disputas, uma terceira parte precisa ter acesso à mensagem, à assinatura digital e a chave privada para checar as três características da assinatura digital.
- ***Arbitrados:*** o documento assinado é enviado a um arbitro que checa as três características fundamentais da assinatura digital, satisfazendo todas o arbitro então envia o documento assinado ao destino garantindo que o mesmo atende aos requisitos necessários. Os métodos de criptografia normalmente envolvidos neste tipo de processo é o de chave pública

Os algoritmos de chave pública tradicional (como o RSA) e de assinatura digital que utilizam estes algoritmos (como o DSA) utilizam chaves de muitos bits. Em 1985 Victor Miller e Neal Koblitz propuseram o uso de sistemas criptográficos baseados em curvas elípticas (ECC), pois estas curvas apresentam a vantagem de utilizarem chaves de menores tamanhos provendo o mesmo nível de segurança. Com chaves de menor tamanho, os algoritmos são mais ágeis, consomem menos espaço para armazenamento, sendo indicados para equipamentos com poucos recursos. A Figura 1 foi obtida em http://www.certicom.com/index.php?action=res,ecc_faq, acessado em 03/07/2006, ela apresenta uma comparação entre os tamanhos de chaves necessários em sistemas baseados no RSA e em curvas elípticas, para termos o mesmo nível de segurança.



NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Figura 1. Tamanhos de chaves

De forma bem simplista podemos dizer que, os algoritmos baseados em curvas elípticas utilizam curvas semelhantes às mostradas na Figura 2. Cada número inteiro é associado a um par de ordenadas (x, y), isto é, um ponto pertencente à curva. Os pontos a serem utilizados são obtidos em um grupo finito. Este grupo finito é formado a partir de um ponto gerador (***G***) pertencente à curva selecionada. O 1º ponto do grupo é definido como sendo ***G***. O 2º ponto do grupo é obtido a partir da adição de ***G*** com ***G***, ou seja, $\boldsymbol{P_2 = G + G = 2\,G}$. E assim por diante.

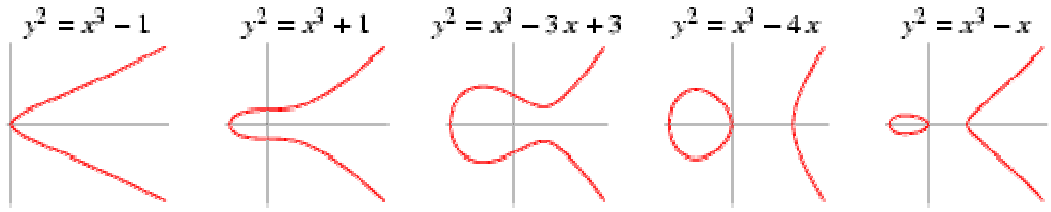


Figura 2. Curvas elípticas

As curvas elípticas usuais são obtidas de uma equação cúbica que pode ser expressa da seguinte forma: $\boldsymbol{y^2 + a\,x\,y + b\,y = x^3 + c\,x^2 + d\,x + e}$. Onde ***a, b, c, d*** e ***e*** são números reais. Uma descrição destas curvas e dos grupos finitos gerados por elas, que são utilizados nos sistemas criptográficos pode ser vista nas referências [1, 3]. Além da questão das chaves serem menores, nos grupos finitos formados pelos pontos de uma curva elíptica, as operações de multiplicação e exponenciação do RSA são equivalentes a operações de soma e multiplicação (somas sucessivas), respectivamente, nas ECC. Isto faz com que os algoritmos baseados em curvas elípticas sejam mais rápidos.

O algoritmo de assinatura digital com curvas elípticas (ECDSA - Elliptic Curve Digital Signature Algorithm) é equivalente ao algoritmo de assinaturas digitais (DSA – Digital Signature Algorithm), porém utilizando ECC. Ele foi proposto inicialmente por Scott Vanstone em 1992 para o NIST (National Institute of Standard and Technology). Ele foi aceito em 1998 pelo ISO (International Standards Organization) [ISO 14888-3], aceito em 1999 pela ANSI (American National Standards Institute) [ANSI X69.2] e em 2000 pelo IEEE (Institue of Eletrical and Eletronic Engineers) [IEEE P1363]. Como o ECDSA foi aceito por tantas instituições de padronização, podemos assumir que ele está estável e é bastante robusto.

Para assinar digitalmente uma mensagem *m*, faça o seguinte na origem:

- Escolha um grupo finito *E* sobre uma curva elíptica.
- Escolha um ponto *G*, gerador do grupo finito $E(Z_p)$ de ordem *n*.
- Escolha uma chave privada *d*, entre 1 e $n - 1$.
- Calcule a chave pública: $Q = d\,G$. Também são públicos: *E*, *n* e *G*.
- Escolha *k* entre 1 e $n - 1$. Calcule $(x_1, y_1) = k\,G$ e $r = x_1$. Volte à escolha de *k* enquanto $r = 0$.

- Calcule $s = k^{-1} [SHA\text{-}1(m) + d\ r] \bmod n$. Volte à escolha de k enquanto $s = 0$.
- O par (r, s) é a assinatura digital de m.

Note que *SHA-1* denota o uso da função *hash* de 160 bits. No destino da mensagem m, para verificar a assinatura digital (r, s) de m, siga os seguintes procedimentos:

- Verifique se r e s são inteiros entre 1 e $n - 1$.
- Calcule $w = s^{-1} \bmod n$.
- Calcule $u_1 = SHA\text{-}1(m)w \bmod n$.
- Calcule $u_2 = rw \bmod n$.
- Calcule $(x_2, y_2) = u_1 G + u_2 Q$ e $v = x_2 \bmod n$.
- Verifique se $v = r$ então a assinatura e a mensagem podem ser aceitas.

Para facilitar o entendimento do algoritmo ECDSA, vamos fazer uma breve descrição dele de forma comparativa com o algoritmo DSA. A Tabela 1 abaixo apresenta esta descrição dos dois algoritmos mostrando o comportamento de cada um deles nas fases: geração das chaves pública e privada, assinatura da mensagem m e verificação da assinatura com a mensagem. A tabela foi construída com base nas informações disponíveis nas referências [2:314, 4 e 5].

Fase	**Algoritmo DSA**	**Algoritmo ECDSA**
Geração das chaves	Escolha p, q, x e $q \mid p - 1$, $1 \le x < q$. Escolha $h \in Z_p^*$ e calcule: $g = h^{(p-1)/q} \bmod p$ até que $g \neq 1$. $y = g^x \bmod p$ Chave pública: (p, q, g, y) Chave privada: x	Escolha E *sobre* Z_p, Escolha d, $1 \le d < n$. Escolha $G \in E(Z_p)$ de ordem n. $Q = d\ G$ Chave pública: (E, n, G, Q) Chave privada: d
Assinatura	Escolha k, $1 \le k < q$. Calcule: $r = (g^k \bmod p) \bmod q$ $s = k^{-1}(h(m) + xr) \bmod q$ (r, s) é a assinatura de m.	Escolha k, $1 \le k < n$. Calcule: $k\ G = (x_1, y_1)$ e $r = x_1 \bmod n$ $s = k^{-1}(h(m) + dr) \bmod n$ (r, s) é a assinatura de m.
Checagem da assinatura	$w = s^{-1} \bmod q$ $u_1 = h(m)w \bmod q$ $u_2 = rw \bmod q$ $v = (g^{u1}\ y^{u2} \bmod p) \bmod q$ Se $v = r$ então (r, s) está ok.	$w = s^{-1} \bmod n$ $u_1 = h(m)w \bmod n$ $u_2 = rw \bmod n$ $u_1 G + u_2 Q = (x_2, y_2)$, $v = x_2 \bmod n$ Se $v = r$ então (r, s) está ok.

Tabela 1. Algoritmo DSA x Algoritmo ECDSA

2.1. Vantagens do ECDSA

O ECDSA apresenta as seguintes vantagens sobre o DSA e outros algoritmos de assinatura digital baseados em no RSA:

- Utiliza chaves menores.
- Utiliza soma ao invés de multiplicações.
- Utiliza multiplicações (soma cumulativa) ao invés de exponenciações.
- É mais eficiente em termos de tempo de resposta.
- Consome menos espaço de armazenamento.
- É adequado para equipamentos com poucos recursos de hardware disponíveis.

2.2. Ataques ao ECDSA

De um modo geral o ECDSA vem se mostrando muito bom no que tange à resistência a ataques, conforme descrito por seus propositores em [5], especialmente para ataques da mensagem conhecida. Entretanto, como todo algoritmo criptográfico ele está sujeito aos seguintes tipos de ataques:

- Ataques contra as curvas elípticas e contra o problema dos logaritmos discretos que fundamentam os criptosistemas baseados nestas curvas.
- Ataques à função *hash* utilizada (SHA-1).
- Outros tipos de ataques.

Dentre os ataques às curvas elípticas e aos logaritmos discretos podemos destacar os seguintes:

- Procura ingênua de d: aplicação da força bruta para quebrar $Q = d\,G$.
- Fatoração da ordem do grupo finito (n) em torno de G: redução da complexidade da fatoração de n utilizando o teorema chinês do resto.
- Curvas definidas sobre grupos finitos pequenos.

Uma lista completa dos ataques pode ser obtida no artigo dos autores, conforme referência [5]. O importante é que, com os parâmetros propostos pela ANSI X9.62, uma máquina com 330 mil processadores precisaria de 32 dias para quebrar um ECC. Assim, o ECDSA é recomendado pelo NIST como um dos algoritmos mais seguros de assinatura digital.

3. Aplicações

O método descrito é utilizado para assinaturas digitais em substituição ao DSA. A razão desta substituição deve-se às vantagens descritas na seção anterior, especialmente em termos de eficácia e eficiência. Por sua vez, as assinaturas digitais podem ser utilizadas para:

- Assinatura irrefutável de mensagens de correio eletrônico, isto acabaria com os spams (pois os emissores poderiam ser responsabilizados pelos seus atos) e permitiria que os e-mails fossem utilizados como provas em processos judiciais.
- Assinatura de pedidos em comércio eletrônico.
- Identificação de sites na Internet, comprovando a autenticidade do endereço acessado.
- Acompanhamento e aditamento de processos judiciais ou administrativos através de meios eletrônicos.
- Recepção do Diário Oficial Eletrônico da União.
- Viabilizar a apresentação de projetos de lei diretamente pelos cidadãos, pois estes podem assinar digitalmente sua adesão ao projeto.
- Assinatura da declaração do IR (Imposto de Renda) e outros serviços da Receita Federal.
- Enviar e receber documentos eletrônicos para cartórios.
- Permitir transações seguras entre instituições financeiras, como por exemplo o SPB – Sistema de Pagamentos Brasileiro, em uso desde 2002.
- Assinatura eletrônica de documentos em geral, em substituição ao enorme número de senhas que as pessoas passaram a ter que criar e gerir com o uso cada vez mais crescente da informática e dos problemas decorrentes de segurança.
- Etc.

4. Implementação Java

Construímos uma aplicação exemplo chamada ECDSA para demonstrar o uso do método proposto e das bibliotecas “java.security”, fornecidas pela SUN, que contém a implementação básica do método citado.

Apesar de constar na documentação da SUN a disponibilidade do algoritmo ECDSA, na prática, esta implementação ainda não está disponível ou a documentação não corresponde à implementação real. Para resolver esta questão tivemos que instalar o provider fornecido pela Bouncy Castle (www.bouncycastle.org). A instalação deste provider foi feita através dos seguintes passos:

1. Entrar no site http://www.bouncycastle.org/latest_releases.html e baixar o arquivo: bcprov-jdk15-133.jar ou mais recente.

2. Copiar este arquivo para as pastas:

   C:\Arquivos de programas\Java\jdk<r>\jre\lib

   C:\Arquivos de programas\Java\jre<r>\lib

   Onde: <r> = release = por exemplo: 1.5.0_07

3. Na seqüência de providers dos arquivos:

   C:\Arquivos de programas\Java\jdk<r>\jre\lib\security\java.security

   C:\Arquivos de programas\Java\jre<r>\lib\security\java.security

   Acrescentar a linha:

   security.provider.<n>=org.bouncycastle.jce.provider.BouncyCastleProvider

   Onde: <n> = número do último provider instalado + 1

4. Instalar a biblioteca bcprov-jdk15-133 apontando para este arquivo numa das pastas do item 2.

   Nossa aplicação possui duas classes:

- ***ECDSA***: Abre a aplicação e chama a classe
- ***ECDSAWindow***: Classe que exibe uma janela contendo um formulário mostrado na Figura 3, onde o usuário preenche o “Texto Plano” e clica no botão “Assinar”. A aplicação gera e mostra um par de chaves (privada e pública) e calcula a assinatura digital do “Texto Plano” fornecido. Se estivéssemos num ambiente cliente – servidor, o servidor deveria enviar para o cliente a mensagem (Texto Plano), a chave pública e a assinatura digital. Finalmente, a aplicação verifica o “Texto Plano” e assinatura digital, utilizando somente a chave pública, simulando o comportamento de um cliente.

Figura 3. Tela da aplicação

A aplicação desenvolvida é bem simples e tem apenas o objetivo didático de detalhar o uso das facilidades oferecidas pela biblioteca fornecida pela SUN.

Nas subseções seguintes vamos apresentar detalhes da implementação da aplicação exemplo, especificamente partes da classe ECDSAWindow.

4.1. Geração de chaves

No início da classe devemos colocar o comando abaixo para importar as definições, os métodos, variáveis, constantes, etc. disponíveis nas bibliotecas.

```
import java.security.*;
import org.bouncycastle.crypto.params.ECPublicKeyParameters;
import org.bouncycastle.jce.ECNamedCurveTable;
import org.bouncycastle.jce.spec.ECNamedCurveParameterSpec;
```

A instrução a seguir define o parâmetro que será utilizado na geração da curva elíptica utilizada pelo algoritmo.

```
ECNamedCurveParameterSpec ECCparam =
        ECNamedCurveTable.getParameterSpec("prime192v2");
```

A instrução abaixo obtém uma instância do algoritmo desejado (ECDSA), provido pela Bouncy Castle, da classe KeyPairGenerator.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("ECDSA", "BC");
```

A instrução abaixo prepara o gerador de chaves para gerar uma chave com o parâmetro definido para a curva elíptica. Esta instrução deve ser utilizada antes de chamar a instrução de geração do par de chaves em si. O algoritmo gera um grupo finito de ordem *n*, escolhe um gerador *G* e gera a chave privada *d*. Finalmente, o algoritmo calcula a chave pública $Q = d\,G$.

```
keyGen.initialize(ECCparam);
keyPair = keyGen.generateKeyPair();
```

As instruções abaixo devolvem respectivamente a chave pública e a chave privada a partir do objeto que contém o par de chaves.

```
PublicKey  pubKey = keyPair.getPublic();
PrivateKey prvKey = keyPair.getPrivate();
```

As instruções abaixo convertem o valor das chaves para o formato de strings e coloca estes valores nas respectivas caixas de texto do formulário.

```
jTACPrivada.setText(prvKey.toString());
jTACPublica.setText(pubKey.toString());
```

4.2. Assinatura digital

A instrução abaixo obtém uma instância do algoritmo ECDSA, da Bouncy Castle, e associa esta instância à um objeto do tipo assinatura.

```
Signature ecdsa = Signature.getInstance("ECDSA", "BC");
```

Entretanto, este objeto precisa ser iniciado com a chave privada para que a assinatura digital possa ser calculada. A instrução abaixo faz isto.

```
ecdsa.initSign(prvKey);
```

Agora basta passar a mensagem (texto plano), na forma de um vetor de bytes, para que a instância do algoritmo gere a assinatura digital correspondente. A instrução a seguir passa o texto para a instância do algoritmo.

```
ecdsa.update(m.getBytes());
```

Para obter a assinatura digital devemos chamar a instrução abaixo. Lembre-se que a assinatura digital também é um vetor de bytes.

```
byte[] sig = ecdsa.sign();
```

A instrução abaixo transforma a assinatura digital para a forma de uma string e mostra a assinatura num campo do formulário.

```
jTFAssinatura.setText(sig.toString());
```

4.3. Verificação da assinatura digital

Para que a instância do algoritmo possa verificar a assinatura digital devemos fornecer para ele a chave pública e a mensagem original (texto plano) como mostrado nas instruções abaixo.

```
ecdsa.initVerify(pubKey);
ecdsa.update(m.getBytes());
```

A instrução abaixo compara a assinatura fornecida com a assinatura calculada a partir do texto plano com a chave pública fornecida. O retorno é do tipo booleano.

```
boolean sigok = ecdsa.verify(sig);
```

Na aplicação verificamos se a assinatura está “Ok” ou não, o que é mostrado como um “Erro”.

5. Conclusão

As curvas elípticas vêm ganhando cada vez mais espaço no mundo dos sistemas criptográficos. Isto se deve à eficiência oferecida por elas em relação aos métodos tradicionais utilizados nos sistemas criptográficos.

O ECDSA, neste momento, pode ser utilizado com segurança como um método de assinatura digital, pois ele foi reconhecido por todas as instituições mais importantes de padronização e recomendação de padrões do mundo.

O trabalho nos permitiu fazer uma revisão teórica do conceito de assinatura digital, das curvas elípticas, dos algoritmos DSA e ECDSA. Também nos permitiu ver quão prática e simples é a implementação, de métodos complexos, quando utilizamos uma biblioteca reconhecidamente de boa qualidade e segura. Tivemos dificuldade de encontrar a versão 5 da J2SE, pois até a versão 1.4 da JCE (antecessora da J2SE) as curvas elípticas não eram suportadas pela biblioteca da SUN. Mesmo assim, nesta versão a implementação ainda não condiz com a documentação. Razão pela qual tivemos que recorrer à implementação da Bouncy Castle.

Acreditamos que o trabalho cumpriu seu propósito de trazer conceitos abstratos da criptografia para um ambiente concreto e prático.

6. Referências

1 João N. Souza, Márcio A. R. Moreira e Ilmério R. Silva, *A Multi-User Key and Data Exchange Protocol to Manage a Secure Database*, SBBD 2002 - XVII Simpósio Brasileiro de Banco de Dados, este artigo está disponível para download em: http://200.146.233.98/empresa/marcio/2002_10_sbbd.pdf.

2 W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 1999.

3 Elliptic Curve, http://mathworld.wolfram.com/EllipticCurve.html

4 D. Johnson e A. Menezes, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, http://citeseer.ist.psu.edu/cache/papers/cs/8755/http:zSzzSzcacr.math.uwaterloo.cazSz~ajmenezezSzpublicationszSzecdsa.pdf/johnson99elliptic.pdf

5 Don Johnson, Alfred Menezes e Scott Vanstone, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, Certicom Corporation, 2001, disponível em http://www.certicom.com/index.php?action=forms,login2&mode=do&next_aid=27&qual=2&