**Capítulo 4**

**Criptografia Assimétrica**

Neste capítulo, você será introduzido aos conceitos e aspectos práticos da criptografia de chave pública, também chamada de criptografia assimétrica ou criptografia de chave assimétrica. Começaremos com os fundamentos teóricos da criptografia de chave pública e construiremos gradualmente os conceitos, com exercícios práticos relevantes usando o OpenSSL. Consulte o capítulo anterior para um guia de instalação do OpenSSL, caso ainda não tenha feito isso. Depois disso, apresentaremos algumas construções criptográficas novas e avançadas.

Ao longo do caminho, discutiremos os seguintes tópicos:

* Matemática fundamental
* Criptografia assimétrica
* Introdução ao RSA
* Introdução à criptografia de curva elíptica (ECC)
* Assinaturas digitais
* Construções criptográficas e tecnologia blockchain

Antes de discutir mais sobre criptografia, alguns termos e conceitos matemáticos precisam ser explicados para construir uma base que permita entender completamente o material apresentado posteriormente neste capítulo.

**Matemática fundamental**

Como o tema da criptografia é baseado em matemática, esta seção apresentará alguns conceitos básicos que o ajudarão a entender os conceitos apresentados adiante. Uma explicação com demonstrações e antecedentes relevantes para todos esses termos exigiria uma matemática um tanto complexa, o que está além do escopo deste livro. Mais detalhes sobre esses tópicos podem ser encontrados em qualquer livro padrão de teoria dos números, álgebra ou criptografia:

* **Aritmética modular**: Também conhecida como aritmética do relógio, os números na aritmética modular “dão a volta” quando atingem um certo número fixo. Esse número fixo é um número positivo chamado de módulo (às vezes abreviado como "mod"), e todas as operações são realizadas com respeito a esse número fixo. Em outras palavras, esse tipo de aritmética lida com os restos após a operação de divisão. Por exemplo, 50 mod 11 é 6 porque 50 dividido por 11 deixa resto 6.
* **Conjuntos**: São coleções de objetos distintos, por exemplo, X = {1, 2, 3, 4, 5}.
* **Grupos**: Um grupo é um conjunto comutativo com uma operação que combina dois elementos do conjunto. A operação do grupo é fechada e associada a um elemento identidade definido. Além disso, cada elemento no conjunto possui um inverso. Fechamento (fechado) significa que se, por exemplo, os elementos A e B estão no conjunto, então o elemento resultante após a realização de uma operação sobre os elementos também está no conjunto. Associatividade significa que o agrupamento dos elementos não afeta o resultado da operação. Quatro axiomas do grupo devem ser satisfeitos para que um conjunto seja considerado um grupo. Esses axiomas do grupo incluem fechamento, associatividade, um elemento identidade e um elemento inverso.
* **Grupo cíclico**: Um tipo de grupo que pode ser gerado por um único elemento chamado gerador do grupo.
* **Corpos**: Um corpo é um conjunto no qual todos os seus elementos formam um grupo aditivo ou multiplicativo. Ele satisfaz axiomas específicos para adição e multiplicação. Para todas as operações do grupo, a lei distributiva também é aplicada. Essa lei determina que a mesma soma ou produto será produzida, mesmo que quaisquer dos termos ou fatores sejam reordenados:
  + Um **corpo finito** é aquele com um conjunto finito de elementos. Também conhecidos como campos de Galois, essas estruturas são de particular importância na criptografia, pois podem ser usadas para produzir resultados exatos e livres de erros nas operações aritméticas.
  + Um **corpo primo** é um corpo finito com um número primo de elementos. Ele tem regras específicas para adição e multiplicação, e cada elemento não nulo no corpo possui um inverso. As operações de adição e multiplicação são realizadas mod p, ou seja, módulo de um número primo.
* **Ordem**: O número de elementos em um corpo. Também é conhecido como a cardinalidade do corpo.

Isto conclui uma introdução básica a alguns conceitos matemáticos envolvidos na criptografia. Na próxima seção, você será introduzido aos conceitos de criptografia.

**Criptografia assimétrica**

Criptografia assimétrica refere-se a um tipo de criptografia onde a chave utilizada para criptografar os dados é diferente da chave utilizada para descriptografar os dados. Essas chaves são chamadas de chave pública e chave privada, respectivamente, razão pela qual a criptografia assimétrica também é conhecida como criptografia de chave pública. Ela utiliza tanto chaves públicas quanto privadas para criptografar e descriptografar dados, respectivamente. Vários esquemas de criptografia assimétrica estão em uso, incluindo a criptografia RSA e ElGamal.

**Chaves públicas e privadas**

Uma chave privada, como o nome sugere, é um número gerado aleatoriamente que é mantido em segredo e guardado privadamente por seus usuários. As chaves privadas precisam ser protegidas e nenhum acesso não autorizado deve ser concedido a essas chaves; caso contrário, todo o esquema da criptografia de chave pública estará comprometido, pois essa é a chave usada para descriptografar mensagens. As chaves privadas podem ter vários tamanhos, dependendo do tipo e da classe de algoritmos utilizados. Por exemplo, no RSA, tipicamente utiliza-se uma chave de 1.024 bits ou 2.048 bits. O tamanho de chave de 1.024 bits não é mais considerado seguro, e recomenda-se pelo menos 2.048 bits.

Uma chave pública está livremente disponível e é publicada pelo proprietário da chave privada. Qualquer pessoa que deseje enviar uma mensagem criptografada ao publicador da chave pública pode fazê-lo criptografando a mensagem usando a chave pública publicada e enviando-a ao detentor da chave privada. Ninguém mais pode descriptografar a mensagem porque a chave privada correspondente está guardada com segurança pelo destinatário pretendido. Uma vez que a mensagem criptografada com a chave pública é recebida, o destinatário pode descriptografá-la usando a chave privada. No entanto, existem algumas preocupações quanto às chaves públicas. Estas incluem a autenticidade e a identificação do publicador das chaves públicas.

Uma representação genérica da criptografia de chave pública é mostrada no diagrama a seguir:

**Figura 4.1: Criptografia/descriptografia usando chaves pública/privada**

O diagrama anterior ilustra como um remetente criptografa os dados P usando a chave pública do destinatário e uma função de criptografia, e produz como saída os dados criptografados C, os quais são então transmitidos pela rede até o receptor. Uma vez que atinge o receptor, ele pode ser descriptografado usando a chave privada do receptor, alimentando os dados criptografados C na função de descriptografia, que produzirá como saída o texto simples P. Dessa forma, a chave privada permanece do lado do receptor, e não há necessidade de compartilhar chaves para realizar a criptografia e a descriptografia, o que é o caso com a criptografia simétrica.

O diagrama a seguir mostra como o receptor utiliza a criptografia de chave pública para verificar a integridade da mensagem recebida. Neste modelo, o remetente assina os dados usando sua chave privada e transmite a mensagem ao receptor. Uma vez que a mensagem é recebida, ela é verificada quanto à integridade usando a chave pública do remetente.

Vale observar que não há criptografia sendo realizada neste modelo. Ele é apresentado aqui apenas para ajudá-lo a compreender completamente o material sobre autenticação e validação de mensagens que será fornecido mais adiante neste capítulo:

**Figura 4.2: Modelo de esquema de assinatura digital com criptografia de chave pública**

O diagrama anterior mostra que o remetente assina digitalmente o texto simples P com sua chave privada usando a função de assinatura S, e produz os dados C, os quais são enviados ao receptor, que verifica C usando a chave pública do remetente e a função V para garantir que a mensagem de fato tenha vindo do remetente.

Os mecanismos de segurança oferecidos pelos criptossistemas de chave pública incluem estabelecimento de chave, assinaturas digitais, identificação, criptografia e descriptografia.

Mecanismos de estabelecimento de chave dizem respeito ao projeto de protocolos que permitem a configuração de chaves sobre um canal inseguro. Serviços de não repúdio (definidos no Capítulo 3, Criptografia Simétrica, como a garantia de que uma ação, uma vez tomada por alguém, não pode ser negada posteriormente), uma opção muito desejável em muitos cenários, podem ser fornecidos usando assinaturas digitais. Às vezes, é importante não apenas autenticar um usuário, mas também identificar a entidade envolvida em uma transação. Isso também pode ser alcançado por uma combinação de assinaturas digitais e protocolos de desafio-resposta. Finalmente, o mecanismo de criptografia para fornecer confidencialidade também pode ser obtido usando criptossistemas de chave pública, como RSA, ECC e ElGamal.

**Algoritmos de criptografia assimétrica**

Algoritmos de chave pública são mais lentos em termos de computação do que algoritmos de chave simétrica. Portanto, eles não são comumente utilizados na criptografia de arquivos grandes ou dos dados reais que precisam ser criptografados. Eles são geralmente usados para troca de chaves para algoritmos simétricos. Uma vez que as chaves são estabelecidas com segurança, algoritmos de chave simétrica podem ser usados para criptografar os dados.

Algoritmos de criptografia de chave pública são baseados em várias funções matemáticas subjacentes. As três principais categorias de algoritmos assimétricos são descritas a seguir.

**Fatoração de inteiros**

Esquemas de fatoração de inteiros são baseados no problema difícil de que inteiros grandes são extremamente difíceis de fatorar. O RSA é um exemplo clássico desse tipo de algoritmo.

**Logaritmo discreto**

Um esquema de logaritmo discreto é baseado em um problema de aritmética modular. É fácil calcular o resultado de uma função módulo, mas é computacionalmente impraticável encontrar o expoente do gerador. Em outras palavras, é extremamente difícil encontrar a entrada a partir do resultado (saída). Isso é chamado de função unidirecional.

Por exemplo, considere a seguinte equação:

32 mod 10 = 9

Agora, dado 9, o resultado da equação anterior, encontrar 2, que é o expoente do gerador (3) na equação, é extremamente difícil de determinar. Esse problema difícil é comumente usado nos algoritmos de troca de chaves de Diffie-Hellman e de assinatura digital.

**Curvas elípticas**

O algoritmo de curva elíptica é baseado no problema do logaritmo discreto discutido anteriormente, mas no contexto de curvas elípticas. Uma curva elíptica é uma curva algébrica cúbica sobre um corpo, que pode ser definida por uma equação, como mostrado aqui. A curva é não singular, o que significa que ela não possui cúspides nem auto-interseções. Ela possui duas variáveis, a e b, bem como um ponto no infinito:

**y² = x³ + ax + b**

Aqui, **a** e **b** são inteiros cujos valores são elementos do corpo sobre o qual a curva elíptica é definida. Curvas elípticas podem ser definidas sobre números reais, racionais, complexos ou corpos finitos. Para fins criptográficos, uma curva elíptica sobre corpos finitos primos é usada em vez de números reais. Além disso, o primo deve ser maior que 3. Diferentes curvas podem ser geradas variando os valores de **a** e/ou **b**.

Os criptossistemas mais proeminentemente utilizados com base em curvas elípticas são o Algoritmo de Assinatura Digital com Curva Elíptica (ECDSA) e a Troca de Chaves de Diffie-Hellman com Curva Elíptica (ECDH).

**Esquema de Criptografia Integrado**

Um esquema de criptografia integrado (IES — *Integrated Encryption Scheme*) é um mecanismo de criptografia híbrido que combina esquemas de chave pública com esquemas de chave simétrica para alcançar conveniência e eficiência. Esquemas de chave pública são convenientes, pois não há necessidade de seguir o processo trabalhoso de compartilhamento de chaves secretas. Por outro lado, esquemas de chave simétrica são mais eficientes do que esquemas de chave pública para criptografia de dados. Assim, esquemas de criptografia híbridos combinam o melhor dos dois mundos para alcançar eficiência e conveniência.

Esquemas híbridos são compostos por dois mecanismos: primeiro, um mecanismo de encapsulamento de chave, que é um criptossistema de chave pública; e segundo, um mecanismo de encapsulamento de dados, que é um mecanismo de criptografia de chave simétrica. Protocolos como TLS e SSH utilizam um esquema de criptografia híbrido. Um IES possui duas variantes: um esquema de criptografia integrado baseado em logaritmo discreto (DLIES) e um esquema de criptografia integrado com curva elíptica (ECIES).

Nas seções a seguir, apresentaremos dois exemplos de criptografia de chave assimétrica: RSA e ECC. O RSA é a primeira implementação de criptografia de chave pública, enquanto a ECC é amplamente usada na tecnologia blockchain.

**Introdução ao RSA**

O RSA foi inventado em 1977 por Ron Rivest, Adi Shamir e Leonard Adelman, daí o nome RSA. Esse tipo de criptografia de chave pública é baseado no problema de fatoração de inteiros, onde a multiplicação de dois números primos grandes é fácil, mas é difícil fatorar o produto (o resultado da multiplicação) de volta nos dois números originais.

O cerne do trabalho envolvido com o algoritmo RSA ocorre durante o processo de geração de chaves. Um par de chaves RSA é gerado realizando os seguintes passos:

1. **Geração do módulo:**
   * Selecione **p** e **q**, que são números primos muito grandes.
   * Multiplique **p** e **q**, **n = p × q**, para gerar o módulo **n**.
2. **Gerar o coprimo:**
   * Assuma um número chamado **e**.
   * **e** deve satisfazer uma certa condição; isto é, deve ser maior que 1 e menor que (p−1)(q−1). Em outras palavras, **e** deve ser um número tal que nenhum número além de 1 possa dividi-lo juntamente com (p−1)(q−1). Isto é chamado de coprimo, ou seja, **e** é o coprimo de (p−1)(q−1).
3. **Gerar a chave pública:**
   * O módulo gerado no passo 1 e o coprimo **e** gerado no passo 2 formam um par que é a chave pública. Essa parte é a parte pública que pode ser compartilhada com qualquer um; no entanto, **p** e **q** devem ser mantidos em segredo.
4. **Gerar a chave privada:**
   * A chave privada é chamada de **d** aqui e é calculada a partir de **p**, **q** e **e**. A chave privada é basicamente o inverso de **e** módulo (p−1)(q−1). Como equação, isso é o seguinte:

**ed ≡ 1 mod (p−1)(q−1)**

Normalmente, um algoritmo de Euclides estendido é usado para tomar **p**, **q** e **e** e calcular **d**. A ideia chave nesse esquema é que qualquer pessoa que conheça **p** e **q** pode facilmente calcular a chave privada **d** aplicando o algoritmo de Euclides estendido. No entanto, alguém que não conheça o valor de **p** e **q** não pode gerar **d**. Isso também implica que **p** e **q** devem ser grandes o suficiente para que o módulo **n** se torne extremamente difícil (computacionalmente impraticável) de fatorar.

Agora, vejamos como as operações de criptografia e descriptografia são realizadas usando RSA. O RSA usa a seguinte equação para produzir o texto cifrado:

**C = Pᵉ mod n**

Isso significa que o texto simples **P** é elevado à potência de **e** e então reduzido módulo **n**. A descriptografia no RSA é fornecida pela seguinte equação:

**P = Cᵈ mod n**

Isso significa que o receptor que tem o par de chaves (n, e) pode decifrar os dados elevando **C** ao valor da chave privada **d**, e então reduzindo módulo **n**.

O artigo original do RSA está disponível aqui:  
<http://web.mit.edu/6.857/OldStuff/Fall03/ref/rivest78method.pdf>  
Rivest, R.L., Shamir, A., and Adleman, L., 1978. *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM, 21(2), pp.120–126.

**Criptografando e descriptografando com RSA**

O exemplo a seguir ilustra como pares de chaves públicas e privadas RSA podem ser gerados usando a linha de comando do OpenSSL.

Primeiro, a chave privada RSA pode ser gerada com o OpenSSL da seguinte forma:

$ openssl genpkey -algorithm RSA -out privatekey.pem -pkeyopt rsa\_keygen\_bits:1024

(Saída esperada com sinais de progresso, por exemplo: ++++++)

Após executar o comando, um arquivo chamado privatekey.pem é produzido, o qual contém a chave privada gerada, conforme abaixo:

$ cat privatekey.pem

-----BEGIN PRIVATE KEY-----

MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAKJOFBzPy2vOd6em...

-----END PRIVATE KEY-----

Como a chave privada está matematicamente ligada à chave pública, também é possível gerar ou derivar a chave pública a partir da chave privada. Utilizando o exemplo da chave privada anterior, a chave pública pode ser gerada da seguinte forma:

$ openssl rsa -pubout -in privatekey.pem -out publickey.pem

writing RSA key

É aceitável revelar a chave privada aqui porque estamos simplesmente usando-a como exemplo.  
No entanto, em sistemas de produção, proteger a chave privada é de extrema importância.  
Certifique-se de que a chave privada esteja sempre em segredo. Além disso, lembre-se de que a chave acima está sendo usada apenas como exemplo; **não a reutilize**.

A chave pública pode ser visualizada com um leitor de arquivos ou qualquer visualizador de texto:

$ cat publickey.pem

-----BEGIN PUBLIC KEY-----

MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCiThQcz8trznenpgZP1Bq8w8u0...

-----END PUBLIC KEY-----

Para ver mais detalhes dos vários componentes, como o módulo, os números primos usados no processo de criptografia, ou expoentes e coeficientes da chave privada gerada, o seguinte comando pode ser utilizado (apenas parte da saída é mostrada, pois a saída real é muito longa):

$ openssl rsa -text -in privatekey.pem

Private-Key: (1024 bit)

modulus:

...

Da mesma forma, a chave pública pode ser inspecionada usando o seguinte comando. As chaves públicas e privadas estão codificadas em base64:

$ openssl pkey -in publickey.pem -pubin -text

-----BEGIN PUBLIC KEY-----

...

-----END PUBLIC KEY-----

Public-Key: (1024 bit)

Modulus:

00:a2:4e:14:1c:cf:cb:6b:ce:77:a7:a6:06:4f:d4:...

Exponent: 65537 (0x10001)

Agora, a chave pública pode ser compartilhada livremente, e qualquer pessoa que deseje enviar-lhe uma mensagem pode usar a chave pública para criptografar a mensagem e enviá-la a você.

Utilizando a chave privada que geramos no exemplo anterior, o comando para criptografar um arquivo de texto message.txt pode ser construído da seguinte maneira:

$ echo datatoencrypt > message.txt

$ openssl rsautl -encrypt -inkey publickey.pem -pubin -in message.txt -out message.rsa

Isso produzirá um arquivo chamado message.rsa, que está em formato binário. Se você exibir message.rsa, verá alguns dados embaralhados, conforme mostrado abaixo:

$ cat message.rsa

s???c?ngJ[Lt!?LgC\[f?L?1?^q?r? a??????Da?=??m??\_?P?Y???KE

Para descriptografar o arquivo criptografado com RSA, o seguinte comando usará a chave privada correspondente para descriptografar o arquivo:

$ openssl rsautl -decrypt -inkey privatekey.pem -in message.rsa -out message.dec

Agora, se o arquivo for lido com cat, o texto simples descriptografado poderá ser visto, como demonstrado aqui:

$ cat message.dec

Datatoencrypt

Com este exemplo, concluímos nossa introdução ao RSA. A seguir, vamos explorar a criptografia de curvas elípticas.

**Introdução à ECC**

ECC é baseada no problema do logaritmo discreto, fundamentado em curvas elípticas sobre corpos finitos (campos de Galois). O principal benefício da ECC sobre outros tipos de algoritmos de chave pública é que ela requer um tamanho de chave menor, fornecendo o mesmo nível de segurança que, por exemplo, o RSA. Dois esquemas notáveis que se originam da ECC são o **ECDH** para troca de chaves e o **ECDSA** para assinaturas digitais.

A ECC também pode ser usada para criptografia, mas isso não é geralmente feito na prática. Em vez disso, a troca de chaves e as assinaturas digitais são mais comumente utilizadas. Como a ECC requer menos espaço para operar, ela está se tornando muito popular em plataformas embarcadas e em sistemas onde os recursos de armazenamento são limitados. Por comparação, o mesmo nível de segurança pode ser alcançado com ECC usando apenas operandos de 256 bits, em comparação com 3.072 bits no RSA.

**Matemática por trás da ECC**

Para entender a ECC, é necessária uma introdução básica à matemática subjacente. Uma curva elíptica é, basicamente, um tipo de equação polinomial conhecida como equação de Weierstrass, que gera uma curva sobre um corpo finito. O campo mais comumente utilizado é aquele onde todas as operações aritméticas são realizadas módulo de um número primo **p**. Os grupos de curva elíptica consistem em pontos sobre a curva em um corpo finito.

Uma curva elíptica é definida pela seguinte equação:

**y² = x³ + Ax + B mod p**

Aqui, **A** e **B** pertencem a um corpo finito ℤₚ ou 𝔽ₚ (um corpo finito primo), juntamente com um valor especial chamado ponto no infinito. O ponto no infinito, ∞, é usado para fornecer operações de identidade para pontos na curva.

Além disso, uma condição também precisa ser satisfeita para garantir que a equação mencionada anteriormente não tenha raízes repetidas. Isso significa que a curva é **não singular**.

A condição é descrita na seguinte equação, que é um requisito padrão que precisa ser atendido. Mais precisamente, isso assegura que a curva seja não singular:

**4a³ + 27b² ≠ 0 mod p**

Para construir o problema do logaritmo discreto baseado em curvas elípticas, é necessário um grupo cíclico suficientemente grande. Primeiro, os elementos do grupo são identificados como um conjunto de pontos que satisfazem a equação anterior. Depois disso, operações de grupo precisam ser definidas sobre esses pontos.

As operações básicas de grupo em curvas elípticas são **adição de pontos** e **duplicação de pontos**. Adição de pontos é um processo no qual dois pontos diferentes são somados, e duplicação de pontos significa que o mesmo ponto é somado a si mesmo.

**Adição de pontos**

A adição de pontos é mostrada no diagrama a seguir. Esta é uma representação geométrica da adição de pontos em curvas elípticas. Nesse método, uma linha diagonal é traçada através da curva que a intersecta em dois pontos, mostrados abaixo como P e Q, o que gera um terceiro ponto entre a curva e a linha.

Esse ponto é refletido como **P + Q**, o que representa o resultado da adição como **R**. Isso é mostrado como **P + Q** no diagrama a seguir:

<imagem\_4.3>

**Figura 4.3: Adição de pontos sobre R**

A operação de grupo denotada pelo sinal **+** para adição resulta na seguinte equação:

**P + Q = R**

Mais precisamente, isso significa que as coordenadas são somadas, como mostrado na equação a seguir:

**(x₁, y₁) + (x₂, y₂) = (x₃, y₃)**

Assim, se **P ≠ Q**, então significa adição de pontos. Para somar esses pontos, veja a seguinte equação:

**s = (y₂ − y₁)(x₂ − x₁)⁻¹ mod p**

**s** representa a inclinação da linha que passa por **P** e **Q**.

Em seguida:

**x₃ = s² − x₁ − x₂ mod p**  
  **y₃ = s(x₁ − x₃) − y₁ mod p**

Até agora, vimos como a operação de adição de pontos funciona e observamos as expressões analíticas da operação de grupo de adição.

A seguir, vejamos um exemplo completo de adição de pontos. Este exemplo mostra a adição e a solução para a equação sobre um corpo finito F₂₃, que contém 23 elementos.

<imagem\_4.4>

**Figura 4.4: Exemplo de adição de pontos**

No exemplo anterior, o gráfico à esquerda mostra os pontos que satisfazem esta equação:

**y² = x³ + 7x + 11**

Há 27 soluções para a equação mostrada acima sobre o corpo finito **F₂₃**. **P** e **Q** são escolhidos para serem somados para produzir o ponto **R**. Os cálculos são mostrados no lado direito, que calcula o terceiro ponto **R**. Note que no gráfico anterior, no cálculo mostrado à direita, **l** é usado para representar a linha que passa por **P** e **Q**.

Como exemplo, para mostrar como a equação é satisfeita pelos pontos mostrados no gráfico, escolhe-se um ponto (x, y) onde **x = 3** e **y = 6**. Este ponto pode ser visualizado no gráfico mostrado aqui. Observe o ponto na coordenada (3, 6), indicado por uma seta:

<imagem\_4.5>

**Figura 4.5: Ponto em (3, 6) mostrado com uma seta**

Usando esses valores, mostramos que a equação é de fato satisfeita:

**y² mod 23 = x³ + 7x + 11 mod 23**  
  **6² mod 23 = 3³ + 7(3) + 11 mod 23**  
  **36 mod 23 = 59 mod 23**  
  **13 = 13**

A próxima seção introduz o conceito de duplicação de pontos, que é outra operação que pode ser realizada em curvas elípticas.

**Duplicação de pontos**

A outra operação de grupo em curvas elípticas é chamada de **duplicação de pontos**. Esse é um processo onde **P** é somado a si mesmo. Este é o caso em que **P** e **Q** estão no mesmo ponto, então, efetivamente, a operação se torna somar o ponto consigo mesmo e é, portanto, chamada de duplicação de ponto. Nesse método, uma **reta tangente** é traçada através da curva, como mostrado no gráfico a seguir. O segundo ponto é obtido na interseção da linha tangente traçada com a curva. Esse ponto é então refletido para gerar o resultado, que é mostrado como **2P = P + P**:

<imagem\_4.6>

**Figura 4.6: Gráfico representando duplicação de ponto**

No caso da duplicação de ponto, a equação torna-se:

**s = (3x₁² + a) / (2y₁) mod p**

**x₃ = s² − x₁ − x₂ mod p**  
  **y₃ = s(x₁ − x₃) − y₁ mod p**

Aqui, **s** é a inclinação da linha tangente que passa por **P**, que é a linha mostrada no topo no gráfico anterior. No exemplo anterior, a curva é traçada como um exemplo simples, e nenhuma solução para a equação é mostrada.

O exemplo a seguir mostra as soluções e a duplicação de ponto de curvas elípticas sobre o corpo finito **F₂₃**.

<imagem\_4.7>

**Figura 4.7: Exemplo de duplicação de ponto**

O gráfico à esquerda mostra os pontos que satisfazem a equação:

**y² = x³ + 7x + 11**

Como mostrado à direita do gráfico anterior, o cálculo encontra **R**, depois que **P** é somado a si mesmo (**duplicação de ponto**). Não há **Q** mostrado aqui porque o mesmo ponto **P** é usado para duplicação.

Usando adição e duplicação, podemos construir outra operação chamada **multiplicação de pontos**, que introduziremos a seguir.

**Multiplicação de pontos**

Essa operação também é chamada de **multiplicação escalar de ponto**. Ela é usada para multiplicar pontos em curvas elípticas por um dado inteiro. Vamos chamar esse inteiro de **d** e o ponto de **P**. Obtemos **dP** somando repetidamente **P**, **d** vezes. Essa operação pode ser descrita da seguinte forma:

**P + P + ... + P = dP**, onde **P** é somado **d** vezes.

Qualquer ponto na curva pode ser somado múltiplas vezes a si mesmo. O resultado dessa operação é sempre outro ponto sobre a curva.

A operação de adição não é eficiente para valores grandes de **d**. No entanto, a operação de multiplicação de pontos pode ser tornada eficiente utilizando o algoritmo de **duplicação e adição** (*double and add*), que combina as operações de adição de ponto e duplicação para obter ganho exponencial de desempenho.

Por exemplo, se utilizarmos apenas adição, para obter **9P**, teríamos que fazer:

**P + P + P + P + P + P + P + P + P**

O que se torna inviável rapidamente à medida que o número de somas de **P** aumenta. Para resolver isso, podemos usar o mecanismo de duplicação-e-adição, no qual primeiro convertemos **9** em binário, depois, começando do bit mais significativo, para cada bit que for **1** (alto), realizamos uma operação de duplicação-e-adição; e para cada **0**, realizamos apenas a operação de duplicação. Não realizamos nenhuma operação no bit mais significativo. Como **9** em binário é **1001**, obtemos para cada bit, da esquerda para a direita: **P**, **2P**, **4P**, e **8P + P**. Esse processo produz **9P** com apenas três operações de duplicação e uma operação de adição, em vez de nove operações de adição.

Neste exemplo, duplicação de ponto e adição foram usadas para construir uma operação eficiente de multiplicação escalar. Agora considere que **dP** resulta em outro ponto sobre a curva; chamaremos esse ponto de **T**. Podemos dizer que, com todas essas duplicações e adições, computamos um ponto chamado **T**. Multiplicamos um ponto **P** sobre a curva por um número **d** para calcular outro ponto **T**.

Aqui está a ideia chave agora: mesmo que conheçamos os pontos **P** e **T**, é computacionalmente inviável reconstruir a sequência de todas as operações de duplicação-e-adição que realizamos para descobrir o número **d**. Em outras palavras, mesmo que alguém conheça **P** e **T**, é quase impossível para essa pessoa encontrar **d**. Isso significa que é uma **função unidirecional**, e é a base do **problema do logaritmo discreto em curvas elípticas (ECDLP)**. Descreveremos o problema do logaritmo discreto em mais detalhes a seguir.

**O problema do logaritmo discreto**

O problema do logaritmo discreto em ECC é baseado na ideia de que, sob certas condições, todos os pontos em uma curva elíptica formam um grupo cíclico.

Em uma curva elíptica, a **chave pública** é um múltiplo aleatório do ponto gerador, enquanto a **chave privada** é um inteiro escolhido aleatoriamente usado para gerar esse múltiplo. Em outras palavras, uma chave privada é um inteiro selecionado aleatoriamente, enquanto a chave pública é um ponto na curva. O problema do logaritmo discreto é o de encontrar a chave privada (um inteiro), onde esse inteiro se encontra entre todos os pontos da curva elíptica. A equação a seguir mostra esse conceito com mais precisão.

Considere uma curva elíptica **E**, com dois elementos **P** e **T**. O problema do logaritmo discreto é encontrar o inteiro **d**, onde **1 ≤ d ≤ #E**, tal que:

**P + P + … + P = dP = T**

Aqui, **T** é a chave pública (um ponto na curva, (x, y)), e **d** é a chave privada. Em outras palavras, a chave pública é um múltiplo aleatório do gerador **P**, enquanto a chave privada é o inteiro usado para gerar esse múltiplo. **#E** representa a ordem da curva elíptica, ou seja, o número de pontos que estão presentes no grupo cíclico da curva elíptica. Um grupo cíclico é formado por uma combinação de pontos da curva elíptica e o ponto no infinito.

O ponto de partida inicial **P** é um parâmetro público, e a chave pública **T** também é publicada, enquanto **d**, a chave privada, é mantida em segredo. Se **d** não for conhecido, é impraticável calculá-lo apenas conhecendo **T** e **P**, o que cria o problema difícil sobre o qual o ECDLP é construído.

Um par de chaves está vinculado aos parâmetros de domínio específicos de uma curva elíptica. Os parâmetros de domínio incluem: o tamanho do campo, a representação do campo, dois elementos dos campos **a** e **b**, dois elementos do campo **Xg** e **Yg**, ordem **n** do ponto **G**, que é calculado como **G = (Xg, Yg)**, e o cofator **h = #E(Fq)/n**. Um exemplo prático usando OpenSSL será descrito mais adiante nesta seção.

Vários parâmetros são recomendados e padronizados para que possam ser usados como curvas com ECC. Um exemplo da especificação **secp256k1** é mostrado aqui. A figura a seguir é um trecho extraído da especificação padrão em <http://www.secg.org/sec2-v2.pdf>. Ela é usada no Bitcoin.

Os parâmetros de domínio da curva elíptica sobre **𝔽ₚ**, associados a uma curva de Koblitz **secp256k1**, são especificados pela sêxtupla **T = (p, a, b, G, n, h)**, onde o corpo finito **𝔽ₚ** é definido por:

**p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFC2F**  
     = 2²⁵⁶ − 2³² − 2⁹ − 2⁸ − 2⁷ − 2⁶ − 2⁴ − 1

A curva **E: y² = x³ + ax + b** sobre **𝔽ₚ** é definida por:

**a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000**  
  **b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007**

O ponto base **G**, em forma comprimida, é:

**G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798**

Na forma não comprimida é:

**G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798  
         483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8**

Finalmente, a ordem **n** de **G** e o cofator são:

**n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141**  
  **h = 01**

Aqui, observamos:

* **p** é o primo que especifica o tamanho do corpo finito.
* **a** e **b** são os coeficientes da equação da curva elíptica.
* **G** é o ponto base que gera o subgrupo requerido, também conhecido como gerador.  
    O ponto base pode ser representado em forma comprimida ou não comprimida.  
    Não é necessário armazenar todos os pontos da curva em uma implementação prática.  
    O gerador comprimido funciona porque os pontos da curva podem ser identificados usando apenas a coordenada **x** e o bit menos significativo de **y**.
* **n** é a ordem do subgrupo.
* **h** é o cofator do subgrupo.

Também podemos usar a linha de comando do OpenSSL para visualizar esses parâmetros da **secp256k1**. Isso pode ser visto aqui:

$ openssl ecparam -param\_enc explicit -text -noout -name secp256k1

Esse comando mostrará a saída a seguir:

**Tipo do campo: campo primo**  
  **Prime:**  
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:  
    ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:ff:  
    ff:fc:2f  
  **A: 0**  
  **B: 7 (0x7)**  
  **Gerador (não comprimido):**  
    04:79:be:66:7e:f9:dc:bb:ac:55:a0:62:95:ce:87:  
    0b:07:02:9b:fc:db:2d:ce:28:d9:59:f2:81:5b:16:  
    f8:17:98:48:3a:da:77:26:a3:c4:65:5d:a4:fb:fc:  
    0e:11:08:a8:fd:17:b4:48:a6:85:54:19:9c:47:d0:  
    8f:fb:10:d4:b8  
  **Ordem:**  
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:  
    ff:fe:ba:ae:dc:e6:af:48:a0:3b:bf:d2:5e:8c:d0:  
    36:41:41  
  **Cofator: 1 (0x1)**

Essa saída pode ser prontamente comparada e verificada com a especificação **SECP256K1** mostrada anteriormente.

Como resumo, uma comparação rápida entre criptografia **RSA** e **ECC** é apresentada abaixo:

| **Característica** | **ECC** | **RSA** |
| --- | --- | --- |
| Tamanho da chave | Menor | Maior |
| Velocidade de geração de chave | Mais rápida | Mais lenta |
| Velocidade de criptografia | Mais rápida | Mais lenta |
| Velocidade de descriptografia | Mais lenta | Mais rápida |
| Resistente a quântica | Não | Não |

Na seção seguinte, exemplos de uso do **OpenSSL** serão mostrados para ajudá-lo a entender os aspectos práticos.

**Gerando chaves com ECC**

O OpenSSL fornece uma biblioteca muito rica de funções para executar ECC. Nesta seção, primeiro será apresentado um exemplo que demonstra a criação de uma chave privada usando as funções ECC disponíveis na biblioteca OpenSSL.

Observe que há diferentes tipos de curvas, e elas devem ser escolhidas com cuidado para garantir garantias de segurança criptográfica.  
Existem também curvas que foram padronizadas pelo NIST nos Estados Unidos e publicadas no documento FIPS 186.

Você pode encontrar mais sobre curvas seguras e seus critérios de seleção em: <https://safecurves.cr.yp.to>

A ECC é baseada em parâmetros de domínio definidos por vários padrões. Você pode ver a lista de todos os padrões disponíveis e as curvas recomendadas no OpenSSL usando o seguinte comando (mais uma vez, apenas parte da saída é mostrada aqui):

$ openssl ecparam -list\_curves

...

brainpoolP384r1: RFC 5639 curve over a 384 bit prime field

brainpoolP384t1: RFC 5639 curve over a 384 bit prime field

brainpoolP512r1: RFC 5639 curve over a 512 bit prime field

brainpoolP512t1: RFC 5639 curve over a 512 bit prime field

Aqui, será gerada uma chave privada baseada em **SECP256K1**:

$ openssl ecparam -name secp256k1 -genkey -noout -out ec-privatekey.pem

Esse comando produzirá um arquivo chamado ec-privatekey.pem, que podemos visualizar com o comando:

$ cat ec-privatekey.pem

-----BEGIN EC PRIVATE KEY-----

MHQCAQEEIJHUIm9NZAgfpUrSxUk/iINq1ghM/ewn/RLNreuR52h/oAcGBSuBBAAK

oUQDQgAE0G33mCZ4PKbg5EtwQjk6ucv9Qc9DTr8JdcGXYGxHdzr0Jt1NInaYE0GG

ChFMT5pK+wfvSLkYl5ul0oczwWKjng==

-----END EC PRIVATE KEY-----

O arquivo ec-privatekey.pem agora contém a chave privada de curva elíptica gerada com base na curva **SECP256K1**. Para gerar uma chave pública a partir da chave privada, execute o seguinte comando:

$ openssl ec -in ec-privatekey.pem -pubout -out ec-pubkey.pem

read EC key

writing EC key

A leitura do arquivo produz a seguinte saída, exibindo a chave pública gerada:

$ cat ec-pubkey.pem

-----BEGIN PUBLIC KEY-----

MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE0G33mCZ4PKbg5EtwQjk6ucv9Qc9DTr8J

dcGXYGxHdzr0Jt1NInaYE0GGChFMT5pK+wfvSLkYl5ul0oczwWKjng==

-----END PUBLIC KEY-----

Agora, o arquivo ec-pubkey.pem contém a chave pública derivada de ec-privatekey.pem. A chave privada pode ser explorada mais detalhadamente com o seguinte comando:

$ openssl ec -in ec-privatekey.pem -text -noout

read EC key

Private-Key: (256 bit)

priv:

00:91:d4:22:6f:4d:64:08:1f:a5:4a:d2:c5:49:3f:

88:83:6a:d6:08:4c:fd:ec:27:fd:12:cd:ad:eb:91:

e7:68:7f

pub:

04:d0:6d:f7:98:26:78:3c:a6:e0:e4:4b:70:42:39:

3a:b9:cb:fd:41:cf:43:4e:bf:09:75:c1:97:60:6c:

47:77:3a:f4:26:dd:4d:22:76:98:13:41:86:0a:11:

4c:4f:9a:4a:fb:07:ef:48:b9:18:97:9b:a5:d2:87:

33:c1:62:a3:9e

ASN1 OID: secp256k1

Da mesma forma, a chave pública pode ser explorada com o seguinte comando:

$ openssl ec -in ec-pubkey.pem -pubin -text -noout

read EC key

Private-Key: (256 bit)

pub:

04:d0:6d:f7:98:26:78:3c:a6:e0:e4:4b:70:42:39:

3a:b9:cb:fd:41:cf:43:4e:bf:09:75:c1:97:60:6c:

47:77:3a:f4:26:dd:4d:22:76:98:13:41:86:0a:11:

4c:4f:9a:4a:fb:07:ef:48:b9:18:97:9b:a5:d2:87:

33:c1:62:a3:9e

ASN1 OID: secp256k1

Também é possível gerar um arquivo com os parâmetros requeridos — neste caso, **SECP256K1** — e então explorá-lo mais detalhadamente para entender os parâmetros subjacentes:

$ openssl ecparam -name secp256k1 -out secp256k1.pem

$ cat secp256k1.pem

-----BEGIN EC PARAMETERS-----

BgUrgQQACg==

-----END EC PARAMETERS-----

O arquivo agora contém todos os parâmetros de SECP256K1, e ele pode ser analisado usando o seguinte comando:

$ openssl ecparam -in secp256k1.pem -text -param\_enc explicit -noout

Esse comando produzirá uma saída semelhante à mostrada abaixo:

**Tipo de Campo:** campo primo  
**Primo:**

00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:

ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:ff:

ff:fc:2f

**A:** 0  
**B:** 7 (0x7)  
**Gerador (não comprimido):**

04:79:be:66:7e:f9:dc:bb:ac:55:a0:62:95:ce:87:

0b:07:02:9b:fc:db:2d:ce:28:d9:59:f2:81:5b:16:

f8:17:98:48:3a:da:77:26:a3:c4:65:5d:a4:fb:fc:

0e:11:08:a8:fd:17:b4:48:a6:85:54:19:9c:47:d0:

8f:fb:10:d4:b8

Ordem:

00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:

ff:fe:ba:ae:dc:e6:af:48:a0:3b:bf:d2:5e:8c:d0:

36:41:41

**Cofator:** 1 (0x1)

O exemplo anterior mostra o número primo usado e os valores de **A** e **B**, junto com o gerador, ordem e cofator dos parâmetros de domínio da curva **SECP256K1**. Com isso, nossa introdução à criptografia de chave pública sob a perspectiva de criptografia e descriptografia está completa.

A seguir, introduziremos as **assinaturas digitais**.

**Assinaturas digitais**

Assinaturas digitais fornecem um meio de associar uma mensagem a uma entidade da qual a mensagem se originou. Assinaturas digitais são usadas para fornecer **autenticação da origem dos dados** e **não repúdio**.

Vários esquemas, como RSA, DSA e ECDSA (baseados em ECC), são usados na prática. O RSA é o mais comum; no entanto, com a crescente adoção da ECC, os esquemas baseados em ECDSA tornaram-se bastante populares. Isso é benéfico nas blockchains porque a ECC fornece o mesmo nível de segurança que o RSA, mas utiliza menos espaço. Além disso, a geração de chaves é muito mais rápida com ECC em comparação ao RSA, o que contribui para o desempenho geral da blockchain. A tabela a seguir mostra que a ECC pode fornecer o mesmo nível de força criptográfica que um sistema baseado em RSA, com um tamanho de chave menor:

| **Tamanho da chave RSA (bits)** | **Tamanho da chave com curva elíptica (bits)** |
| --- | --- |
| 1.024 | 160 |
| 2.048 | 224 |
| 3.072 | 256 |
| 7.680 | 384 |
| 15.360 | 521 |

As assinaturas digitais são calculadas em dois passos. Como exemplo, os passos de alto nível do esquema de assinatura digital RSA são os seguintes:

**Algoritmos de assinatura digital RSA**

Algoritmos de assinatura digital baseados em RSA são calculados utilizando os dois passos listados abaixo. Fundamentalmente, a ideia é primeiro calcular o hash dos dados e então assinar com a chave privada:

* **Calcular o valor hash do pacote de dados**:  
    Isso fornecerá a garantia de integridade dos dados, pois o hash pode ser novamente calculado no lado do receptor e comparado com o hash original para verificar se os dados foram modificados durante a transmissão. Tecnicamente, a assinatura de mensagens pode funcionar sem primeiro aplicar hash aos dados, mas isso **não é considerado seguro**.
* **Assinar o valor hash com a chave privada do signatário**:  
    Como somente o signatário possui a chave privada, a **autenticidade** da assinatura e dos dados assinados é assegurada.

As assinaturas digitais possuem algumas propriedades importantes, como:

* **Autenticidade**: As assinaturas digitais podem ser verificadas por uma parte receptora.
* **Infalsificabilidade**: Garante que apenas o remetente da mensagem pode usar a funcionalidade de assinatura utilizando a chave privada.  
    Assinaturas digitais devem fornecer proteção contra falsificação. Falsificação significa que um adversário fabricaria uma assinatura válida para uma mensagem sem qualquer acesso à chave privada do signatário legítimo.  
    Em outras palavras, infalsificabilidade significa que ninguém mais pode produzir a mensagem assinada gerada por um remetente legítimo. Isso também é chamado de **propriedade de não repúdio**.
* **Não reutilizável**: A assinatura digital não pode ser separada de uma mensagem e reutilizada em outra.  
    Em outras palavras, a assinatura digital está firmemente ligada à mensagem correspondente e não pode ser simplesmente recortada de sua mensagem original e anexada a outra.

O funcionamento genérico de uma função de assinatura digital é mostrado no diagrama a seguir:

<imagem\_4.8>

**Figura 4.8: Assinatura digital (esquerda) e processo de verificação (direita) — exemplo com assinaturas digitais RSA**

Se um remetente deseja enviar uma mensagem autenticada a um receptor, há dois métodos que podem ser usados:

* **Assinar e depois criptografar (Sign then encrypt)**:  
    Com essa abordagem, o remetente assina digitalmente os dados usando a chave privada, anexa a assinatura aos dados, e então criptografa os dados e a assinatura digital usando a chave pública do receptor. Este é considerado um esquema **mais seguro** em comparação ao próximo método.
* **Criptografar e depois assinar (Encrypt then sign)**:  
    Neste método, o remetente criptografa os dados usando a chave pública do receptor e depois assina digitalmente os dados criptografados.

**Gerando assinaturas digitais RSA com OpenSSL**

Assinaturas digitais podem ser facilmente geradas usando **OpenSSL**. Um exemplo prático de geração de uma assinatura digital usando RSA é mostrado a seguir. Os arquivos de chave pública e privada **devem** já estar disponíveis para que a assinatura digital funcione.

Neste exemplo, utilizaremos os arquivos privatekey.pem e publickey.pem que foram gerados anteriormente.

Crie um arquivo de mensagem de texto chamado message.txt com o seguinte conteúdo:

$ echo hello > message.txt

Agora gere um resumo da mensagem (um hash SHA-256, por exemplo) e depois assine esse hash com a chave privada:

$ openssl dgst -sha256 -sign privatekey.pem -out signature.bin message.txt

O comando acima gera um arquivo chamado signature.bin, que contém a assinatura digital.

A assinatura pode ser verificada com a chave pública correspondente. Para isso, execute o seguinte comando:

$ openssl dgst -sha256 -verify publickey.pem -signature signature.bin message.txt

Se a assinatura for válida, a saída será:

Verified OK

Caso contrário, será retornada uma mensagem de erro indicando falha na verificação. Isso pode ocorrer se:

* a assinatura não corresponder à mensagem,
* a chave pública estiver incorreta,
* a mensagem original tiver sido modificada,
* a assinatura tiver sido corrompida ou substituída.

**Esquema de Assinatura Digital com Curvas Elípticas (ECDSA)**

A ECC também pode ser usada para criar assinaturas digitais. O esquema mais utilizado é chamado de **ECDSA** (*Elliptic Curve Digital Signature Algorithm*). O funcionamento é semelhante ao do RSA, mas os cálculos subjacentes utilizam a matemática das curvas elípticas.

O ECDSA segue um processo similar ao RSA:

1. Calcula-se um hash da mensagem a ser assinada.
2. Utiliza-se a chave privada para gerar uma assinatura baseada nesse hash.
3. A assinatura pode ser verificada por qualquer pessoa que tenha a chave pública correspondente.

Como a ECC permite o uso de chaves menores com a mesma segurança, o ECDSA gera assinaturas mais curtas e é mais eficiente em ambientes com recursos limitados, como dispositivos móveis e sistemas embarcados. Além disso, ECDSA é amplamente utilizado em **blockchains**, especialmente no **Bitcoin**, onde é o principal esquema de assinatura digital adotado.

**Assinando e verificando com ECDSA usando OpenSSL**

Agora, veremos como realizar o processo de **assinatura digital** e **verificação** usando **ECDSA** com OpenSSL. Assumiremos que você já possui uma chave privada ECC em um arquivo chamado ec-privatekey.pem, como gerado anteriormente.

Vamos utilizar o mesmo arquivo de mensagem criado anteriormente:

$ echo hello > message.txt

**Assinar a mensagem**

Para assinar o conteúdo do arquivo message.txt com a chave privada ECC, use o seguinte comando:

$ openssl dgst -sha256 -sign ec-privatekey.pem -out ec-signature.bin message.txt

Esse comando irá:

* Calcular o hash SHA-256 do conteúdo de message.txt.
* Assinar o hash usando a chave privada ECC armazenada em ec-privatekey.pem.
* Salvar a assinatura digital resultante em ec-signature.bin.

**Verificar a assinatura**

Para verificar a assinatura digital com a chave pública correspondente armazenada em ec-pubkey.pem, use o seguinte comando:

$ openssl dgst -sha256 -verify ec-pubkey.pem -signature ec-signature.bin message.txt

Se a assinatura for válida, a seguinte mensagem será exibida:

Verified OK

Esse processo demonstra o uso de ECDSA para assinar e verificar mensagens. Na prática, isso é extremamente útil para validar transações, arquivos e outras formas de comunicação seguras onde a **autenticidade** e a **integridade** são cruciais.

**Conclusão**

Neste capítulo, aprendemos que:

* A **criptografia assimétrica** usa um par de chaves: pública e privada.
* Ela resolve os problemas de distribuição de chave da criptografia simétrica.
* Existem diferentes tipos de algoritmos assimétricos, como **RSA**, **algoritmos baseados em logaritmo discreto**, e **curvas elípticas (ECC)**.
* **ECC** fornece a mesma segurança que RSA com tamanhos de chave menores, tornando-a mais eficiente e adequada para sistemas embarcados e blockchains.
* **Assinaturas digitais** garantem autenticidade, integridade e não repúdio, sendo fundamentais em ambientes distribuídos como blockchains.
* Aprendemos a **gerar pares de chaves**, **criptografar/descriptografar dados**, e **assinar/verificar assinaturas digitais** usando **RSA** e **ECC** com **OpenSSL**.