**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**

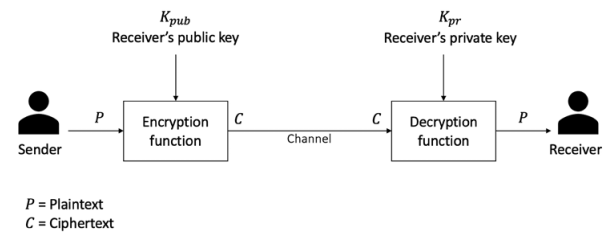
Criptografia assimétrica refere-se a um tipo de criptografia onde a chave utilizada para criptografar os dados é diferente da chave utilizada para descriptografar os dados. Essas chaves são chamadas de chave pública e chave privada, respectivamente, razão pela qual a criptografia assimétrica também é 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:

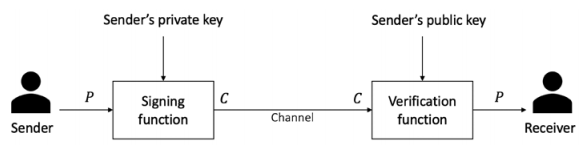


**Figura 4.1: Criptografia/descriptografia usando chaves pública/privada**

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**

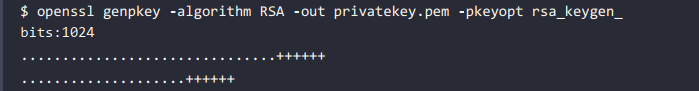
Isso significa que o receptor que tem o par de chaves (n, e) pode decifrar os dados elevando **C** ao valor da chave privada **d**, e então reduzindo módulo **n**.

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.

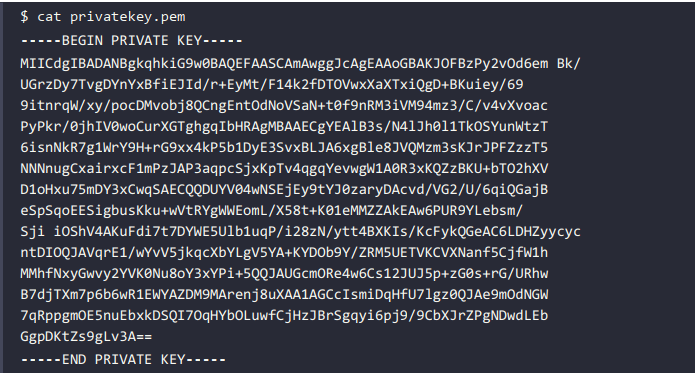
**Criptografando e descriptografando com RSA**

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:

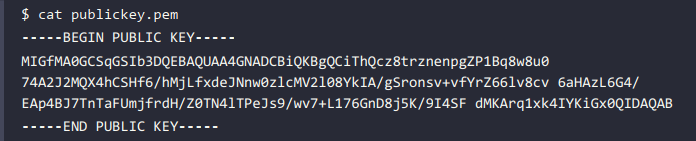
 (Saída esperada com sinais de progresso, por exemplo: ++++++)

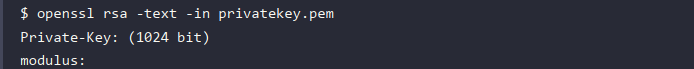
Após executar o comando, um arquivo chamado privatekey.pem é produzido, o qual contém a chave privada gerada, conforme abaixo:

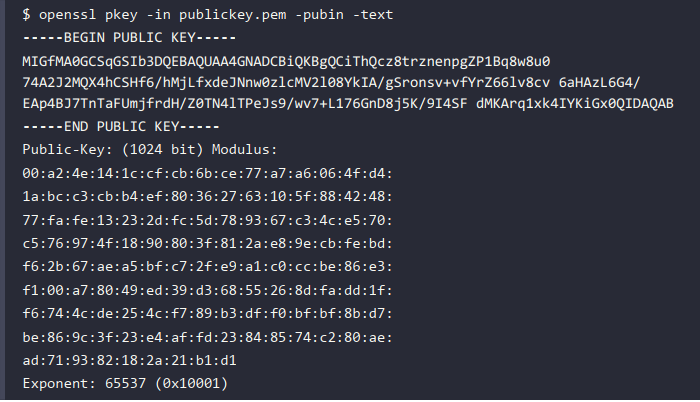
Como a chave privada está matematicamente ligada à chave pública, também é possível gerar ou derivar a chave pública a partir da chave privada. Utilizando o exemplo da chave privada anterior, a chave pública pode ser gerada da seguinte forma:

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:

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):

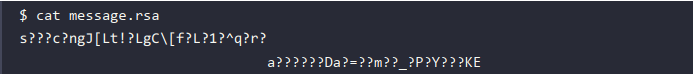
Da mesma forma, a chave pública pode ser inspecionada usando o seguinte comando. As chaves públicas e privadas estão codificadas em base64:

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

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

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

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

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**

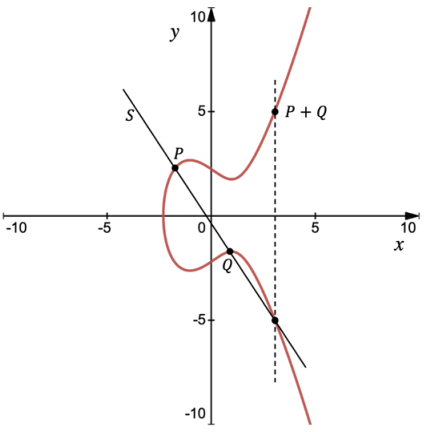
Para construir o problema do logaritmo discreto baseado em curvas elípticas, é necessário um grupo cíclico suficientemente grande. Primeiro, os elementos do grupo são identificados como um conjunto de pontos que satisfazem a equação anterior. Depois disso, operações de grupo precisam ser definidas sobre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



**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**

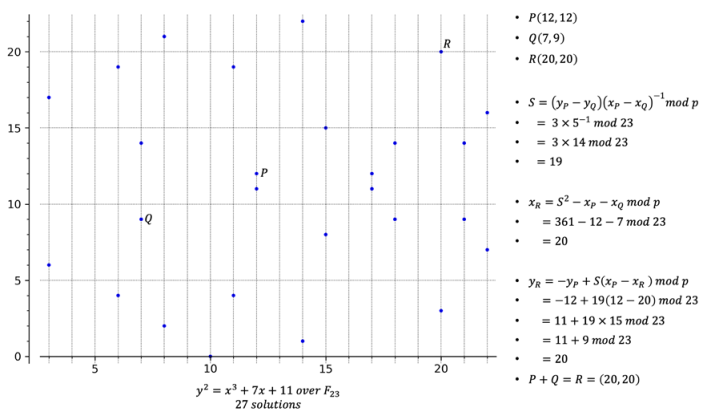
**s** representa a inclinação da linha que passa por **P** e **Q**.

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.

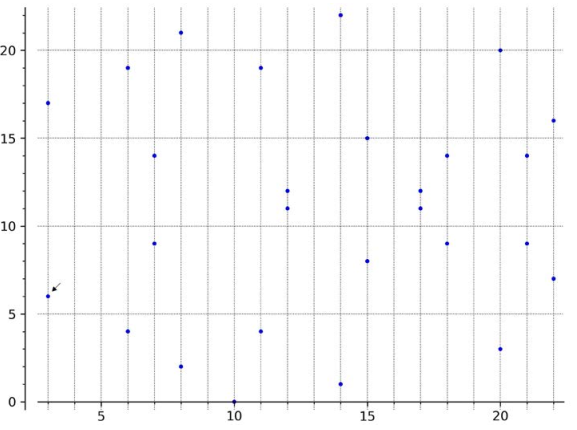
**Figura 4.4: Exemplo de adição de pontos**

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:



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

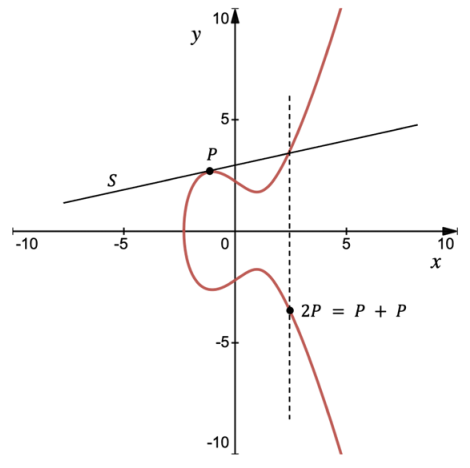
Usando esses valores, mostramos que a equação é de fato satisfeita:

**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**:



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

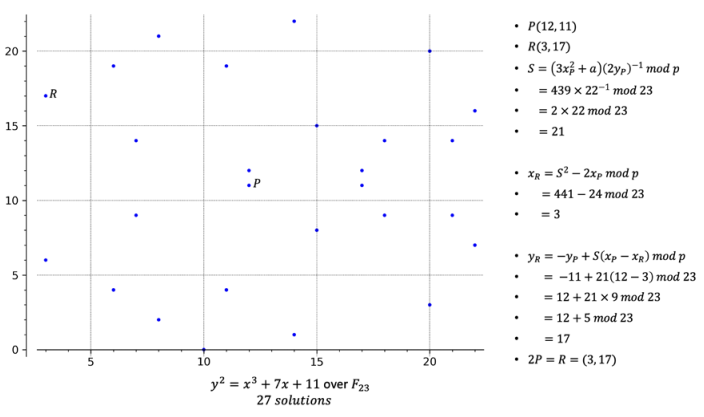
No caso da duplicação de ponto, a equação torna-se:

**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**.

**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**

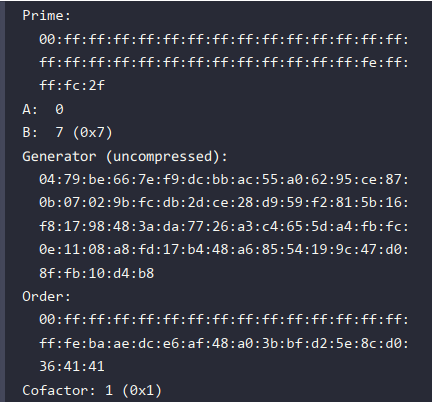
Aqui, observamos:

* **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**  
  ****

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.

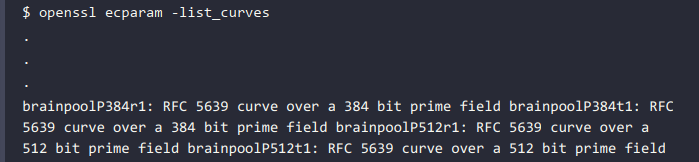
**Gerando chaves com ECC**

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.

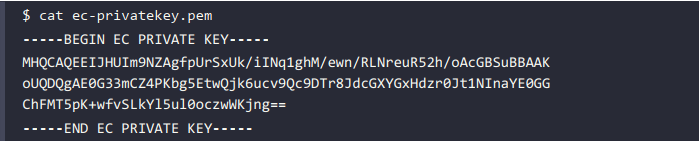
Observe que há diferentes tipos de curvas, e elas devem ser escolhidas com cuidado para garantir garantias de segurança criptográfica.  
Existem também curvas que foram padronizadas pelo NIST nos Estados Unidos e publicadas no documento FIPS 186.

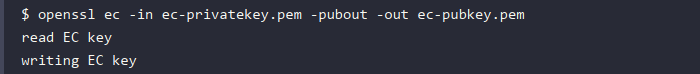
Você pode encontrar mais sobre curvas seguras e seus critérios de seleção em: <https://safecurves.cr.yp.to>

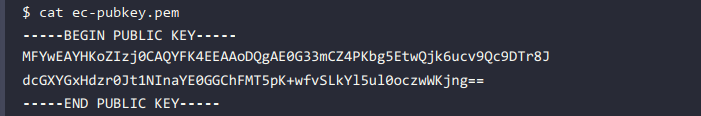
A ECC é baseada em parâmetros de domínio definidos por vários padrões. Você pode ver a lista de todos os padrões disponíveis e as curvas recomendadas no OpenSSL usando o seguinte comando (mais uma vez, apenas parte da saída é mostrada aqui):

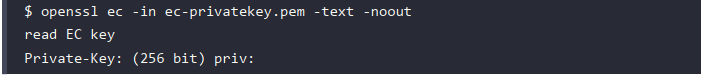
Aqui, será gerada uma chave privada baseada em **SECP256K1**:

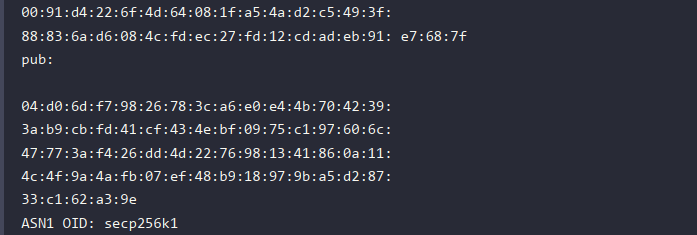
Esse comando produzirá um arquivo chamado ec-privatekey.pem, que podemos visualizar com o comando:

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:

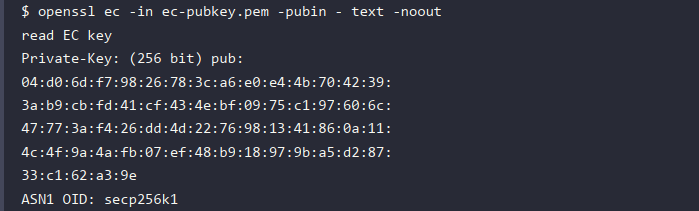
A leitura do arquivo produz a seguinte saída, exibindo a chave pública gerada:

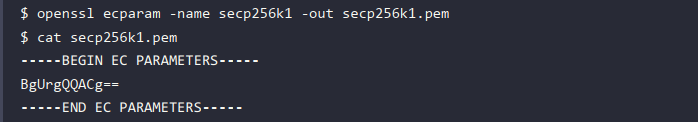
Agora, o arquivo ec-pubkey.pem contém a chave pública derivada de ec-privatekey.pem. A chave privada pode ser explorada mais detalhadamente com o seguinte comando:





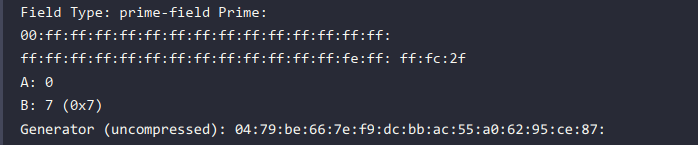
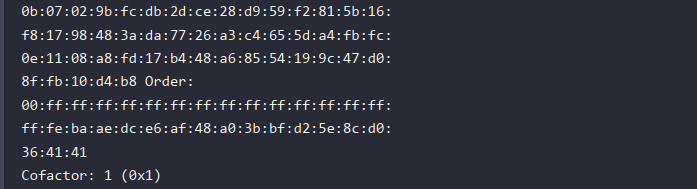
Da mesma forma, a chave pública pode ser explorada com o seguinte comando:

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

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

Esse comando produzirá uma saída semelhante à mostrada abaixo:

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

****

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**

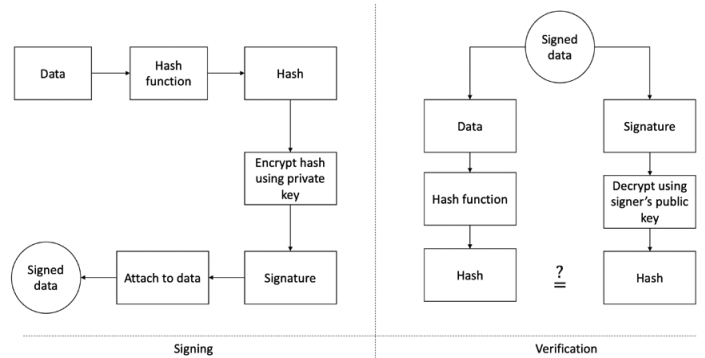
Algoritmos de assinatura digital baseados em RSA são calculados utilizando os dois passos listados abaixo. Fundamentalmente, a ideia é primeiro calcular o hash dos dados e então assinar com a chave privada:

* **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:

**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:

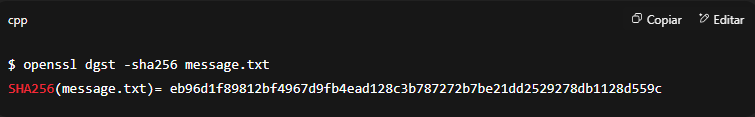
* **Assinar e depois criptografar (Sign then encrypt)**:  
    Com essa abordagem, o remetente assina digitalmente os dados usando a chave privada, anexa a assinatura aos dados, e então criptografa os dados e a assinatura digital usando a chave pública do receptor. Este é considerado um esquema **mais 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.

Na prática, um certificado digital que contém a assinatura digital é emitido por uma autoridade certificadora (AC) que associa uma chave pública a uma identidade

Agora serão mostrados vários exemplos práticos que demonstram como a assinatura digital RSA pode ser gerada, usada e verificada usando o OpenSSL.

**Geração de assinaturas digitais RSA**

O primeiro passo é gerar um hash do arquivo da mensagem. Note que o hash SHA-256 foi escolhido apenas como exemplo:



Tanto a geração do hash quanto a assinatura podem ser feitas em um único passo, como mostrado aqui. Note que o privatekey.pem foi gerado nos passos fornecidos anteriormente:



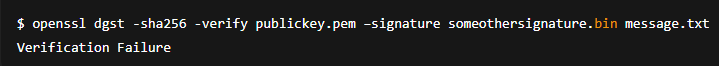
Agora, vamos exibir o diretório mostrando os arquivos relevantes:



Para verificar a assinatura, a seguinte operação pode ser realizada:



Da mesma forma, se algum outro arquivo de assinatura inválido for usado, a verificação falhará, como mostrado aqui:



Agora que entendemos como o esquema de assinatura digital RSA funciona, na próxima seção, vamos introduzir o ECDSA, que é outro esquema popular de assinatura digital.

**O algoritmo de assinatura digital de curva elíptica**  
O ECDSA é um DSA baseado em curvas elípticas. O DSA é um padrão para assinaturas digitais. É baseado em exponenciação modular e no problema do logaritmo discreto. É usado nas plataformas de blockchain do Bitcoin e Ethereum para validar mensagens e fornecer serviços de integridade de dados. Agora, descreveremos como o ECDSA funciona.

Na prática, um certificado digital que contém a assinatura digital é emitido por uma autoridade certificadora (CA) que associa uma chave pública com uma identidade.

**Capítulo 4**

Para assinar e verificar usando o esquema ECDSA, primeiro é necessário gerar um par de chaves:

1. Primeiro, defina uma curva elíptica E com o seguinte:

• Módulo P  
• Coeficientes a e b  
• Ponto gerador A que forma um grupo cíclico de ordem prima q

1. Um número inteiro d é escolhido aleatoriamente tal que 0 < d < q.
2. Calcule a chave pública B tal que B = dA:

• A chave pública é um sêxtuplo na forma mostrada aqui:

Kpb = (p, a, b, q, A, B)

• A chave privada é um número inteiro escolhido aleatoriamente d no passo 2:

Kpr = d

• Agora, a assinatura pode ser gerada usando as chaves privada e pública.  
• Uma chave efêmera Ke é escolhida, onde 0 < Ke < q. Deve-se garantir que Ke seja verdadeiramente aleatória e que nenhuma duas assinaturas usem a mesma chave; caso contrário, a chave privada pode ser calculada.  
• Outro valor R é calculado usando R = KeA—isto é, multiplicando A (o ponto gerador) pela chave efêmera aleatória.  
• Inicialize uma variável r com o valor da coordenada x do ponto R de modo que r = xR.  
• A assinatura pode ser calculada como segue:

S = (h(m) + dr)Ke⁻¹ mod q

1. Aqui, m é a mensagem para a qual a assinatura está sendo computada, e h(m) é o hash da mensagem m.
2. A verificação da assinatura é feita seguindo este processo:

• O valor auxiliar w é calculado como w = s⁻¹ mod q  
• Valor auxiliar u1 = w \* h(m) mod q  
• Valor auxiliar u2 = w \* r mod q  
• Calcule o ponto P, P = u1A + u2B

1. A verificação é realizada como segue:

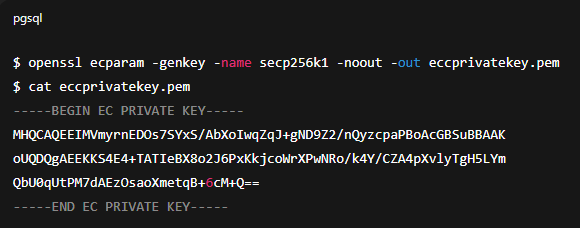
• r, s é aceito como uma assinatura válida se a coordenada x do ponto P calculado no passo 4 tiver o mesmo valor que o parâmetro de assinatura r mod q, ou seja:

Xp = r mod q → assinatura válida  
Xp ≠ r mod q → assinatura inválida

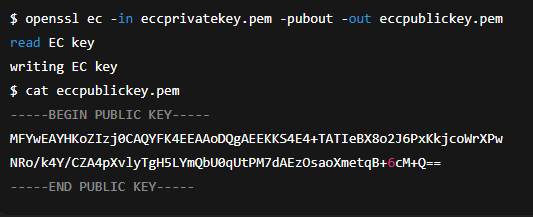
Em seguida, será apresentado um exemplo que mostra como o OpenSSL pode ser usado para realizar operações relacionadas ao ECDSA com base em ECC.

**Criptografia Assimétrica**

**Gerando assinaturas digitais ECDSA**  
Neste exemplo, veremos como uma assinatura digital ECDSA pode ser gerada usando OpenSSL. Primeiro, a chave privada é gerada usando os seguintes comandos:

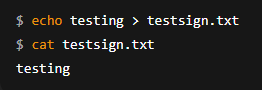


Em seguida, a chave pública é gerada a partir da chave privada:



Agora, suponha que um arquivo chamado testsign.txt precise ser assinado e verificado. Isso pode ser feito da seguinte forma:

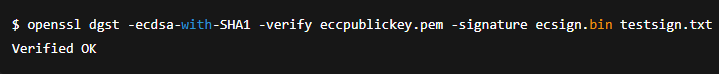
1. Crie um arquivo de teste:



Execute o seguinte comando para gerar uma assinatura usando a chave privada para o arquivo testsign.txt:



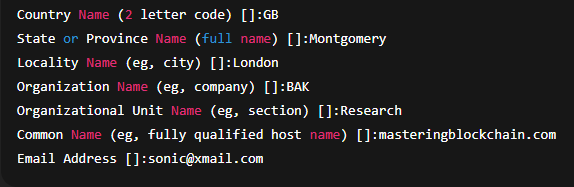
Finalmente, o comando para verificação pode ser executado, como mostrado aqui:



Um certificado também pode ser produzido usando a chave privada gerada anteriormente, utilizando o seguinte comando:



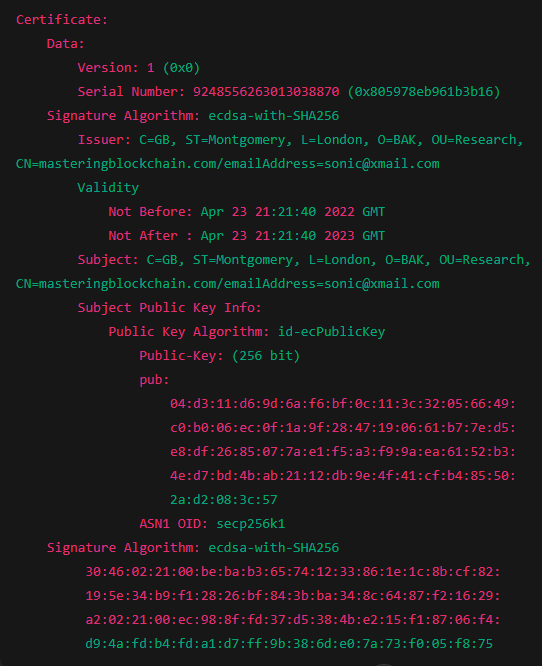
Insira os parâmetros apropriados para gerar o certificado:



Uma vez gerado, o certificado pode ser explorado usando o seguinte comando:



A saída mostra o certificado:



**Diferentes tipos de assinaturas digitais**  
Agora, forneceremos uma visão geral de diferentes tipos de assinaturas digitais, sua relevância e aplicações na tecnologia blockchain. Existem diferentes tipos de esquemas de assinatura digital com base na matemática subjacente que utilizam. Famílias de esquemas de assinatura são:

* Assinaturas e chaves públicas baseadas em RSA
* Assinaturas digitais baseadas em logaritmo discreto

As assinaturas RSA e as chaves públicas têm pelo menos 256 bytes de tamanho, o que não é adequado para blockchains, pois exigem mais espaço. No entanto, elas são rápidas de verificar, mas devido à exigência desnecessária de espaço, não são usadas em blockchains.

Assinaturas de logaritmo discreto são comumente usadas em blockchains. Elas são pequenas, geralmente de 48 ou 64 bytes com chaves públicas de 32 bytes. Esse tamanho pequeno as torna úteis para blockchains, e são usadas no Bitcoin, Ethereum e em várias outras blockchains. Esta família inclui as assinaturas Schnorr e ECDSA.

Assinaturas digitais são usadas em blockchains, onde as transações são assinadas digitalmente pelos remetentes usando sua chave privada antes que o remetente transmita a transação à rede. Essa assinatura digital prova que o remetente é o legítimo proprietário do ativo, por exemplo, dos bitcoins, e está autorizado. Essas transações são verificadas novamente por outros nós na rede para garantir que os fundos pertencem de fato ao nó (usuário) que afirma ser o proprietário. Elas também são usadas em protocolos de consenso para assinar votos, por exemplo, em esquemas de consenso BFT.

Alguns outros tipos, com base na diferença em sua aplicação e mecânica, são descritos abaixo.

**Assinaturas cegas**  
O esquema de assinatura cega foi inventado por David Chaum em 1982. Ele é baseado em esquemas de assinatura digital de chave pública, como RSA. A ideia central por trás das assinaturas cegas é fazer com que a mensagem seja assinada pelo signatário, sem realmente revelar a mensagem. Isso é alcançado ao disfarçar ou cegar a mensagem antes de assiná-la, daí o nome "assinaturas cegas".

Essa assinatura cega pode então ser verificada em relação à mensagem original, assim como uma assinatura digital normal. As assinaturas cegas foram introduzidas como um mecanismo para permitir o desenvolvimento de esquemas de dinheiro digital. Especificamente, uma assinatura cega pode fornecer serviços de não rastreabilidade e anonimato em um sistema distribuído, como uma blockchain.

O esquema de assinatura cega também é usado para construir o Blindcoin, que é usado no protocolo Mixcoin para o Bitcoin, para esconder endereços de usuários do serviço de mixagem. O protocolo Mixcoin permite pagamentos anônimos na rede Bitcoin, mas os endereços dos usuários ainda são visíveis para o serviço de mixagem. O Blindcoin é um protocolo que resolve essa limitação.

**Assinaturas múltiplas**  
No esquema de assinatura múltipla (multisignature), um grupo de entidades assina uma única mensagem. Em outras palavras, várias chaves únicas mantidas por seus respectivos proprietários são usadas para assinar uma única mensagem.

As assinaturas múltiplas também são às vezes chamadas de assinaturas multipartidárias na literatura. Em implementações de blockchain, as assinaturas múltiplas fornecem a capacidade de permitir que transações sejam assinadas por vários usuários, o que resulta em maior segurança. Isso também é chamado de "multi-sig" e foi implementado no Bitcoin. Esses esquemas podem ser usados de tal forma que o número necessário de assinaturas possa ser definido para autorizar transações. Por exemplo, uma assinatura múltipla 1-de-2 pode representar uma conta conjunta onde qualquer um dos dois titulares da conta conjunta pode autorizar uma transação assinando-a.

Em outra variação, por exemplo, uma assinatura múltipla 2-de-2 pode ser usada em um cenário onde ambas as assinaturas dos titulares da conta conjunta são necessárias para assinar a transação. Esse conceito também é generalizado como m de n assinaturas, onde m é o número mínimo de assinaturas necessárias e n é o número total de assinaturas.

O artigo original de David Chaum sobre assinaturas cegas está disponível em:  
<http://blog.koehntopp.de/uploads/Chaum.BlindSigForPayment.1982.PDF>

Chaum, David. “Blind signatures for untraceable payments.” In Advances in cryptology, pp. 199–203. Springer, Boston, MA, 1983.

Mais informações sobre este esquema estão disponíveis no seguinte artigo:

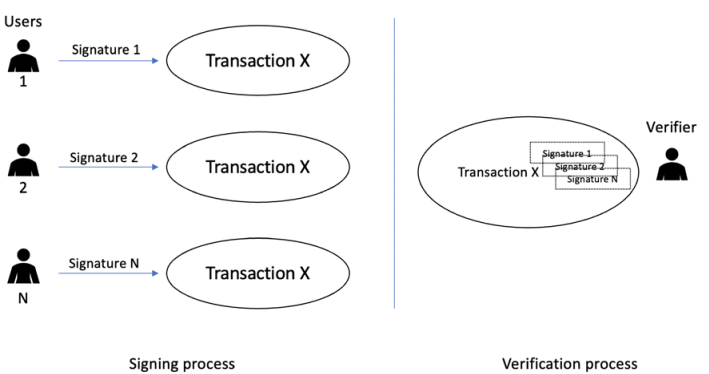
Valenta, L. e Rowan, B., janeiro. “Blindcoin: Blinded, accountable mixes for bitcoin.” In International Conference on Financial Cryptography and Data Security, pp. 112–126. Springer, Berlin, Heidelberg, 2015.

Este esquema foi introduzido em 1983 por Itakura et al. em seu artigo:

“A Public-key Cryptosystem Suitable for Digital Multisignatures, vol. 71.” Nec Research & Development, pp. 474–480, 1983.

**Criptografia Assimétrica**

Este conceito pode ser visualizado com o seguinte diagrama:



*Figure 4.9: Multisignature scheme*

A figura anterior mostra o processo de assinatura no lado esquerdo, onde m é o número de diferentes usuários, e mantendo m assinaturas únicas assinam uma única transação. Quando o validador ou verificador a recebe, todas as assinaturas nela precisam ser verificadas individualmente.

**Assinaturas por limiar (Threshold signatures)**  
Este esquema é um tipo específico de assinatura múltipla. Ele não depende dos usuários para assinarem a mensagem com suas próprias chaves únicas; em vez disso, requer apenas uma chave pública e uma chave privada, e resulta em apenas uma assinatura digital. Em assinaturas múltiplas, a mensagem resultante contém assinaturas digitais de todos os signatários e requer verificação individual pela parte verificadora, mas em assinaturas por limiar, o verificador verifica apenas uma assinatura digital. A ideia central do esquema é dividir a chave privada em várias partes, e cada signatário mantém sua própria parte da chave privada. O processo de assinatura exige que cada usuário use sua respectiva parte da chave privada para assinar a mensagem. Nesse esquema, apenas um subconjunto dos signatários pode produzir a assinatura com sua parte, e não há necessidade de todos os membros colaborarem para produzir a assinatura.

As blockchains Openchain e Multichain fazem uso de esquemas de assinatura múltipla.

Mais informações sobre o Openchain estão disponíveis em:  
<https://docs.openchain.org/en/latest/general/overview.html>

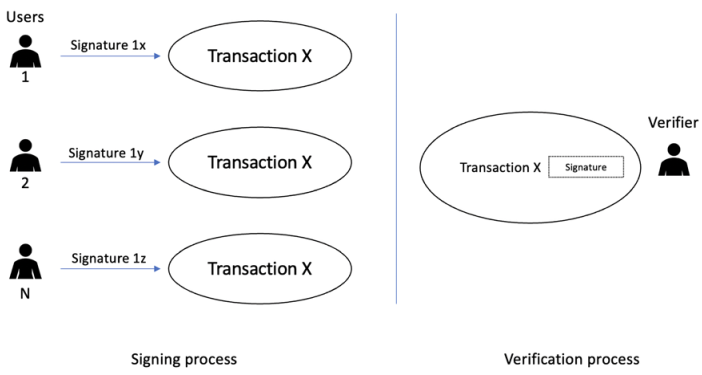
Mais informações sobre o esquema de assinatura múltipla do Multichain estão disponíveis em:  
<https://www.multichain.com/developers/multisignature-transactions/>

A comunicação entre signatários é governada por um protocolo de comunicação específico. Em contraste com as assinaturas múltiplas, as assinaturas por limiar resultam em um tamanho menor da transação e são mais rápidas de verificar. Uma assinatura por limiar aumenta a disponibilidade e resiliência do esquema, pois as partes da assinatura são armazenadas por signatários distintos (servidores).

Uma ligeira limitação, no entanto, é que para que as assinaturas por limiar funcionem, todos os signatários envolvidos no processo de assinatura devem permanecer online para participar de um protocolo interativo para gerar a assinatura, enquanto que nas assinaturas múltiplas, a assinatura pode ser fornecida de forma assíncrona; isto é, os usuários podem fornecer assinaturas sempre que estiverem disponíveis.

Há também variantes que são não-interativas, onde cada usuário computa sua parte da assinatura sem qualquer comunicação online com outros usuários (servidores). Por outro ângulo, pode haver um cenário em esquemas de assinatura múltipla onde um usuário pode reter sua assinatura de forma maliciosa, o que pode resultar em negação de serviço. Com assinaturas por limiar, é possível construir a assinatura exigida apenas com qualquer subconjunto autorizado do grupo total, isto é, ttt de nnn, mas pelo menos esses ttt participantes devem estar online.

As assinaturas por limiar também podem ser usadas para fornecer serviços de anonimato em uma rede blockchain. Isso ocorre porque os esquemas de assinatura por limiar não revelam os membros no grupo de limiar que assinaram para produzir a assinatura. Isso contrasta com os esquemas de assinatura múltipla, que revelam as identidades de todos os signatários. Além disso, em redes permissionadas, as assinaturas por limiar podem ser usadas em cenários onde um limiar de nós (usuários) é necessário para concordar com uma operação.

****

*Figure 4.10: Threshold signature scheme*

A figura anterior mostra o processo de assinatura no lado esquerdo, onde m diferentes usuários, mantendo diferentes partes (fragmentos) de uma assinatura digital, assinam uma única transação. Quando o validador ou verificador a recebe, apenas uma assinatura precisa ser verificada.

**Assinaturas agregadas (Aggregate signatures)**  
Assinaturas agregadas são usadas para reduzir o tamanho de assinaturas digitais. Esse esquema é particularmente útil em cenários onde múltiplas assinaturas digitais estão em uso. A ideia central é agregar múltiplas assinaturas em uma única assinatura, sem aumentar o tamanho da assinatura de uma única mensagem. É simplesmente um tipo de assinatura digital que suporta agregação. A pequena assinatura agregada é suficiente para fornecer verificação ao verificador de que todos os usuários assinaram suas mensagens originais. Assinaturas agregadas são comumente usadas para reduzir o tamanho das mensagens em protocolos de rede e segurança. Por exemplo, o tamanho das cadeias de certificados digitais em uma infraestrutura de chave pública (PKI) pode ser reduzido significativamente ao comprimir todas as assinaturas da cadeia em uma única assinatura. As assinaturas agregadas Boneh–Lynn–Shacham (BLS) são um exemplo popular de assinatura agregada.

A agregação permite comprimir muitas assinaturas de muitos usuários diferentes em uma única assinatura de 48 bytes. Em vez de escrever x assinaturas para x transações, você pode comprimir e escrever apenas uma assinatura na blockchain. Essa propriedade torna as assinaturas BLS muito úteis para blockchains devido ao benefício de economia de espaço. Além disso, são rápidas de verificar, o que as torna ainda mais adequadas para blockchains. As assinaturas BLS permitem que você construa esquemas por limiar do tipo m de n com muita facilidade, onde um limiar de m usuários é necessário para assinar uma transação dentre n usuários.

Assinaturas BLS são usadas no Ethereum e em algumas outras blockchains. As assinaturas BLS têm 48 bytes de comprimento; elas podem ser agregadas e permitem construir esquemas por limiar. A maioria das novas blockchains usa assinaturas BLS devido ao seu tamanho reduzido e à funcionalidade de agregação.

**Assinaturas em anel (Ring signatures)**  
Os esquemas de assinatura em anel permitem um mecanismo onde qualquer membro de um grupo de signatários pode assinar uma mensagem em nome de todo o grupo. O requisito chave aqui é que a identidade do signatário real que assinou a mensagem deve permanecer desconhecida (computacionalmente inviável de determinar) para um observador externo. Parece igualmente provável que qualquer um no grupo confiável de signatários tenha assinado a mensagem, mas não é possível descobrir quem realmente assinou a mensagem. Cada membro do grupo do anel mantém uma chave pública e uma chave privada. As assinaturas em anel podem ser usadas para fornecer serviços de preservação de privacidade (anonimato).  
Uma variação das assinaturas em anel chamada **assinaturas em anel rastreáveis** permite prevenir gastos duplos, e é usada no **CryptoNote**. A criptomoeda **Monero** também usa assinaturas em anel.

Nesta seção, cobrimos diferentes tipos de assinaturas digitais, suas operações e sua relevância para blockchains. A seguir, apresentaremos alguns tópicos mais avançados em criptografia.

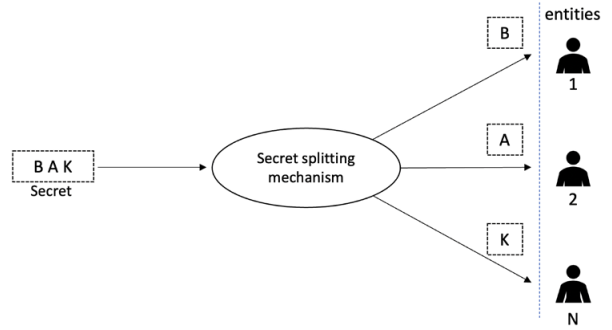
**Construtos criptográficos e tecnologia blockchain**  
Agora, apresentaremos alguns tópicos avançados em criptografia que não são apenas importantes por si só, mas também relevantes para a tecnologia blockchain devido às suas várias aplicações nesse espaço.

**Criptografia homomórfica (Homomorphic encryption)**  
Um algoritmo de criptografia é homomórfico se for capaz de aplicar alguma operação sobre dados criptografados sem precisar descriptografá-los. Criptossistemas de chave pública como o RSA são homomórficos multiplicativos ou aditivos, como o sistema Paillier, e são chamados de **sistemas parcialmente homomórficos**.  
Criptografias parcialmente homomórficas aditivas (PHEs) são adequadas para aplicações de votação eletrônica e bancárias, devido à sua capacidade de realizar operações adicionais sobre dados criptografados. Por exemplo, em votações, elas podem somar votos mesmo que estes estejam criptografados.

Até recentemente, não havia nenhum sistema que suportasse ambas as operações, mas em 2009, um sistema **totalmente homomórfico** foi descoberto por **Craig Gentry**. Como esses esquemas permitem o processamento de dados criptografados sem a necessidade de descriptografia, eles possuem muitas aplicações potenciais, especialmente em cenários onde é necessário manter a privacidade, mas os dados também precisam ser processados por partes potencialmente não confiáveis — por exemplo, computação em nuvem e motores de busca online.

Desenvolvimentos recentes na criptografia homomórfica têm sido bastante promissores, e pesquisadores estão trabalhando ativamente para torná-la eficiente e mais prática. Isso é de particular interesse na tecnologia blockchain, como será descrito mais adiante neste livro, pois pode resolver o problema de confidencialidade e privacidade na blockchain.

**Compartilhamento de segredo (Secret sharing)**  
Compartilhamento de segredo é o mecanismo de distribuir um segredo entre um conjunto de entidades. Todas as entidades dentro de um conjunto recebem uma parte única do segredo após ele ser dividido em várias partes. O segredo pode ser reconstruído combinando todas ou algumas partes (um número ou limiar específico) do segredo. As partes individuais do segredo, por si só, não revelam nada sobre o segredo.

****

**Esquemas de comprometimento (Commitment schemes)**  
Os esquemas de comprometimento são geralmente descritos como um equivalente criptográfico digital de um envelope lacrado. Um comprometimento em si não revela nenhuma informação sobre o valor real dentro dele. Este esquema ocorre em duas fases, a saber:

* **Fase de comprometimento (commit):** a fase de comprometimento oferece duas propriedades de segurança — ou seja, **ocultamento** e **vinculação**:
  + A propriedade de ocultamento garante que um adversário não possa descobrir nenhuma informação nem revelar o valor comprometido antes da fase de abertura.
  + A propriedade de vinculação garante que, uma vez que o remetente tenha se comprometido com um valor, esse valor ou mensagem não possa ser alterado.
* **Fase de abertura (open):** a fase de abertura (também chamada de fase de revelação) é onde o receptor recebe algumas informações adicionais do remetente. Isso permite ao receptor estabelecer o valor oculto (escondido) pelo comprometimento com uma prova de que o remetente não modificou o valor após o comprometimento — ou seja, que o remetente não enganou durante a fase de comprometimento.

Os esquemas de comprometimento desempenham um papel vital na construção de blockchains que preservam a privacidade. Eles têm aplicações na construção de protocolos de conhecimento zero e protocolos de prova de intervalo (*range proofs*). Blockchains como **Monero** e **Zcash** utilizam esquemas de comprometimento para introduzir privacidade.

A ideia central por trás dos esquemas de comprometimento é que eles permitem que alguém se comprometa secretamente com um valor com a possibilidade de revelá-lo mais tarde. Esses esquemas impedem as partes de mudarem o valor ao qual se comprometeram depois que já fizeram o comprometimento. Um exemplo notável de esquema de comprometimento é o **esquema de comprometimento de Pedersen**.

Na próxima seção, introduziremos as **provas de conhecimento zero (Zero-Knowledge Proofs - ZKPs)**, que são um assunto fascinante e uma área muito fértil para pesquisa. Já existem várias blockchains e criptomoedas que utilizam provas de conhecimento zero para fornecer privacidade. Um exemplo importante é o **Zcash** (<https://z.cash>).

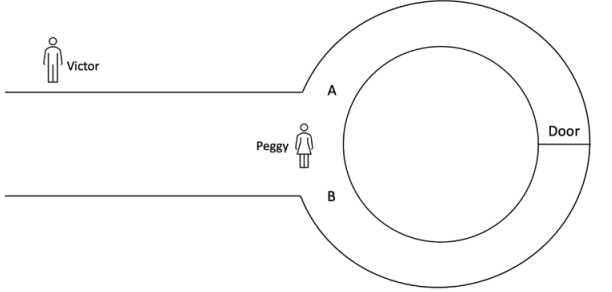
**Provas de conhecimento zero (Zero-knowledge proofs)**  
As ZKPs foram introduzidas por **Goldwasser**, **Micali** e **Rackoff** em 1985. Essas provas são usadas para provar a validade de uma afirmação sem revelar absolutamente nenhuma informação sobre a afirmação. Existem três propriedades das ZKPs que são necessárias: **completude**, **correção (ou solidez)** e a **propriedade de conhecimento zero**:

* **Completude (completeness)** garante que, se uma determinada afirmação for verdadeira, então o verificador será convencido dessa alegação pelo provador.
* A **propriedade de correção (soundness)** assegura que, se uma afirmação for falsa, então nenhum provador desonesto pode convencer o verificador do contrário.
* A **propriedade de conhecimento zero (zero-knowledge)**, como o nome indica, é a propriedade chave das ZKPs, pela qual se garante que absolutamente nada é revelado sobre a afirmação exceto se ela é verdadeira ou falsa.

As ZKPs são **probabilísticas** em vez de determinísticas. A natureza probabilística das ZKPs ocorre porque esses protocolos exigem várias rodadas para alcançar gradualmente um nível mais alto de certeza, a cada rodada, permitindo ao verificador aceitar que o provador de fato conhece o segredo.

Na literatura, geralmente se usa uma analogia conhecida como **Caverna de Ali Baba** para explicar as ZKPs. O diagrama a seguir mostra uma variante dessa analogia de como o protocolo de conhecimento zero funciona. Mostra dois personagens chamados **Peggy** e **Victor**, cujos papéis são de Provador (*Prover*) e Verificador (*Verifier*), respectivamente. Peggy conhece uma palavra mágica secreta para abrir a porta em uma caverna, que é em forma de anel. Ela tem uma entrada dividida e uma porta mágica no meio do anel que bloqueia o outro lado. Também rotulamos as entradas esquerda e direita como **A** e **B**.

O artigo original sobre o esquema de comprometimento de Pedersen está disponível em:  
<https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf>  
Pedersen, T.P., 1991, agosto. **Verifiable Secret Sharing não interativo e teoricamente seguro em termos de informação.** In *Annual international cryptology conference* (pp. 129–140). Springer, Berlin, Heidelberg.

****

*Figure 4.12: Analogy of the zero-knowledge mechanism*

Victor quer saber o segredo, mas Peggy não quer revelá-lo. No entanto, ela gostaria de provar a Victor que realmente conhece o segredo:

Agora, há várias etapas que são tomadas tanto por Peggy quanto por Victor para chegar a uma conclusão sobre se Peggy conhece ou não o segredo:

* Primeiro, Victor espera fora da entrada principal da caverna e Peggy entra na caverna.
* Peggy escolhe aleatoriamente a entrada A ou B da caverna.
* Agora, Victor entra na caverna e grita A ou B aleatoriamente, pedindo para Peggy sair pela saída que ele nomeou.
* Victor registra por qual saída Peggy saiu.

Agora, suponha que Victor pediu para Peggy sair pela saída A e ela saiu pela saída B. Victor então sabe que Peggy **não** conhece o segredo. Se Peggy sair pela saída A, então há **50% de chance** de que ela conheça o segredo — mas isso também significa que ela pode ter tido sorte e escolhido A para entrar na caverna desde o início, e agora apenas voltou sem precisar passar pela porta mágica. No entanto, se esta rotina for repetida várias vezes, e dado que Victor está escolhendo A ou B aleatoriamente, a cada rodada do protocolo, as chances de Peggy ter sorte diminuem. Se Peggy repetidamente conseguir sair pela entrada que Victor nomeou, então é **altamente provável** que ela conheça o segredo para abrir a porta mágica.

Note que durante este processo, Peggy **não revelou o segredo em momento algum**, mas ainda assim conseguiu convencer Victor com alta probabilidade de que ela conhece o segredo.

Lembre-se: dissemos anteriormente que protocolos de conhecimento zero são probabilísticos.  
Peggy e Victor repetindo esse processo provam o conhecimento do segredo com alta probabilidade,  
pois a probabilidade de adivinhação ou sorte se reduz a um ponto desprezível após algumas rodadas.

**As fases de uma prova de conhecimento zero (ZKP)**  
Uma ZKP compreende as seguintes fases:

* **Fase da testemunha (Witness phase):** nesta fase, o provador envia a prova da afirmação ao verificador.
* **Fase de desafio (Challenge phase):** nesta fase, o verificador escolhe uma pergunta (desafio) e a envia ao provador.
* **Fase de resposta (Response phase):** nesta fase, o provador gera uma resposta e a envia como resposta ao verificador. O verificador então verifica a resposta para confirmar se o provador realmente conhece a afirmação.

Esse esquema também pode ser visualizado usando o diagrama a seguir, que mostra como o protocolo de conhecimento zero funciona geralmente e quais etapas o provador e o verificador tomam para executar o protocolo com sucesso:

****Embora conhecimento zero (ZK) não seja um conceito novo, esses protocolos despertaram um interesse especial entre pesquisadores da área de blockchain devido às suas propriedades de privacidade, que são altamente desejáveis em finanças e muitas outras áreas, incluindo direito e medicina. Um exemplo importante de implementação bem-sucedida de um mecanismo ZKP é a criptomoeda **Zcash**. No Zcash, o **zk-SNARK** (*Zero-Knowledge Succinct Non-interactive ARgument of Knowledge*) é implementado para fornecer anonimato e confidencialidade.

As aplicações de ZKP incluem:

* **Prova de posse:** ou seja, prova de que o provador é o dono de algum segredo ou chave privada sem revelar sua natureza.
* **Prova de associação:** o provador pode demonstrar que pertence a uma organização ou grupo sem revelar sua identidade.
* **Prova de idade:** o provador demonstra que tem, por exemplo, mais de 25 anos, mas sem revelar sua idade exata.
* **Pagamento de impostos com privacidade:** cidadãos podem pagar impostos sem revelar sua renda.

**zk-SNARKs**  
Os protocolos de conhecimento zero geralmente são interativos, pois exigem interações repetidas entre o provador e o verificador. No entanto, existem protocolos que não requerem interação entre provador e verificador. Esses tipos de ZKP são chamados de **provas não interativas**. Um exemplo proeminente é o **zk-SNARK**.

Antes da introdução dos zk-SNARKs, as ZKPs eram consideradas pouco práticas devido à complexidade que surge dos requisitos de interações repetidas entre provador e verificador e ao tamanho grande das provas.

Os zk-SNARKs possuem várias propriedades, descritas aqui:

* **zk (Zero-knowledge):** esta propriedade permite que o provador convença o verificador de que uma afirmação é verdadeira sem revelar qualquer informação sobre ela. Uma afirmação neste contexto é qualquer programa de computador que termine e não demore muito para ser executado, isto é, não consuma muitos ciclos.
* **Succinct (S):** esta propriedade permite uma prova de tamanho pequeno que é rápida de verificar.
* **Non-interactive (N):** esta propriedade permite que o provador prove uma afirmação enviando apenas uma única mensagem (prova) ao verificador.
* **ARguments (AR):** estes são os argumentos para convencer o verificador de que a afirmação do provador é verdadeira. Lembre-se de que discutimos **correção (soundness)** e **completude** anteriormente. No caso dos argumentos, o provador é limitado computacionalmente, e é computacionalmente inviável para ele enganar o verificador.
* **Knowledge (K):** esta propriedade significa que o provador tem a evidência de que ele de fato conhece a afirmação que afirma conhecer.

Houve alguns trabalhos muito influentes publicados sobre provas de conhecimento zero não interativas ao longo dos anos.  
O mais proeminente deles, e o que propôs o primeiro design universal, é o artigo de Eli Ben-Sasson et al., disponível em:  
<https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-ben-sasson.pdf>

Ben-Sasson, E., Chiesa, A., Tromer, E. e Virza, M., 2014.  
*Succinct non-interactive zero knowledge for a von Neumann architecture.*  
In 23rd USENIX Security Symposium (USENIX Security 14), pp. 781–796.

Outros trabalhos notáveis anteriores que contribuíram para o desenvolvimento dos zk-SNARKs incluem:

* Gennaro, R., Gentry, C., Parno, B., e Raykova, M., “Quadratic span programs and succinct NIZKs without PCPs.”  
  *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 626–645, Springer, Berlin, Heidelberg, 2013.  
  <https://link.springer.com/content/pdf/10.1007/978-3-642-38348-9_37.pdf>
* Groth, J. e Sahai, A., “Efficient non-interactive proof systems for bilinear groups.”  
  *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 415–432, Springer, Berlin, Heidelberg, 2008.  
  <https://link.springer.com/content/pdf/10.1007/978-3-540-78967-3_24.pdf>

**zk-SNARKs: construção da prova**

Os zk-SNARKs foram implementados em diferentes blockchains. O exemplo mais proeminente é o **Zcash**, que os utiliza em seu recurso de **transações protegidas (shielded transactions)** para fornecer confidencialidade e anonimato.  
O suporte para os primitivos criptográficos exigidos por zk-SNARKs também foi introduzido na blockchain **Ethereum** com a atualização **Byzantium**.

As provas zk-SNARKs são geradas por meio de várias etapas diferentes. Gerar zk-SNARKs para um programa (ou afirmação) **não é um processo simples**, pois exige que o programa seja convertido em um circuito com um formato muito específico. Esse formato específico é chamado de **programa aritmético quadrático (QAP – Quadratic Arithmetic Program)**.  
O processo de geração da prova é o seguinte:

****

*Figure 4.14: zk-SNARK construction*

O processo segue as etapas abaixo:

* **Circuito aritmético:** o primeiro passo na construção do zk-SNARK é converter uma etapa lógica de um cálculo nas menores unidades possíveis compostas por operações aritméticas básicas.  
  As operações aritméticas incluem adição, subtração, multiplicação e divisão. No circuito aritmético, o cálculo é apresentado como fios e portas, representando o fluxo de entradas e as operações aritméticas realizadas sobre essas entradas.  
  É um **grafo acíclico direcionado (DAG)** que avalia um polinômio ao receber entradas e realizar operações aritméticas sobre elas.
* **R1CS:** R1CS é a abreviação de **Rank 1 Constraint System (Sistema de Restrições de Rank 1)**. Basicamente, esse sistema é um conjunto de restrições que permite verificar todas as etapas do circuito aritmético e confirmar que, ao final do processo, a saída é a esperada.
* **QAP:** o provador utiliza os QAPs para construir uma prova da afirmação. No R1CS, o verificador precisa verificar muitas restrições diferentes, mas com uma representação QAP do circuito, todas as diferentes restrições podem ser agrupadas em uma única restrição.
* Por fim, o QAP é usado no protocolo zk-SNARK para provar a afirmação por meio da interação entre provador e verificador.

Os zk-SNARKs **não são uma solução mágica (silver bullet)** para os problemas de privacidade. Em vez disso, como muitas outras tecnologias, eles possuem prós e contras.

A maior crítica aos zk-SNARKs é a **configuração inicial de confiança (trusted setup)**. Essa configuração inicial pode ser influenciada e comprometida. No entanto, se for feita corretamente, ela funciona — mas **sempre há uma chance de que a configuração inicial tenha sido comprometida**, e ninguém jamais será capaz de descobrir.  
Essa é a questão que foi abordada pelos **zk-STARKs**.

**zk-STARKs**  
**Zero-Knowledge Scalable Transparent ARguments of Knowledge (zk-STARKs)** são um novo tipo de ZKP que resolveu várias limitações dos zk-SNARKs.

As principais diferenças entre os esquemas zk-SNARK e zk-STARK estão listadas na tabela a seguir:

| **Propriedades** | **zk-SNARKs** | **zk-STARKs** |
| --- | --- | --- |
| Escalabilidade | Menos escalável | Mais escalável |
| Configuração inicial | Requerida | Não requerida — possui um mecanismo publicamente verificável |
| Resistência pós-quântica | Não resistente a ataques de computadores quânticos | Resistente a ataques de computadores quânticos |
| Técnicas de construção | Baseadas em curvas elípticas e emparelhamentos | Baseadas em funções hash e conceitos da teoria da informação |

Embora esse esquema seja mais eficiente, resistente a ataques quânticos, escalável e transparente,  
a maior limitação está no **tamanho da prova**, que tem **algumas centenas de kilobytes** em comparação aos **288 bytes** dos zk-SNARKs.  
Isso é considerado um problema em blockchains públicas, onde **provas grandes podem afetar a escalabilidade e o desempenho**.

**Provas de intervalo de conhecimento zero (Zero-knowledge range proofs)**  
As **provas de intervalo de conhecimento zero (ZKRPs)** são usadas para provar que um determinado número está dentro de um intervalo específico.  
Isso pode ser útil quando, por exemplo, alguém **não deseja revelar seu salário**, mas está disposto a **revelar apenas o intervalo** dentro do qual o salário se encontra.  
As provas de intervalo também podem ser usadas para **verificações de idade** sem exigir que o provador revele sua **idade exata**.

Nesta seção, cobrimos o tópico muito interessante das **provas de conhecimento zero (ZKPs)** e suas técnicas e desenvolvimentos relevantes.  
As ZKPs constituem uma área **muito ativa de pesquisa**, especialmente no contexto das blockchains, onde essa tecnologia pode **fornecer os tão necessários serviços de privacidade e confidencialidade** em redes blockchain públicas.

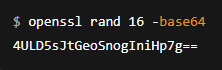
**Esquemas de codificação (Encoding schemes)**  
Além dos primitivos criptográficos, esquemas de codificação **binário-para-texto** também são usados em vários cenários.  
O uso mais comum é converter dados binários em texto para que possam ser processados, armazenados ou transmitidos por meio de um protocolo que **não suporta o processamento de dados binários**.  
Por exemplo, imagens podem ser armazenadas em um banco de dados codificadas em **base64**, o que permite que um campo de texto possa armazenar uma imagem.  
Outra codificação, chamada **base58**, foi popularizada por seu uso no **Bitcoin**.

Este esquema foi projetado por **Eli Ben-Sasson** et al.  
O artigo original sobre o assunto está disponível em:  
<https://eprint.iacr.org/2018/046.pdf>

Ben-Sasson, E., Bentov, I., Horesh, Y., e Riabzev, M.,  
“Scalable, transparent, and post-quantum secure computational integrity.”  
IACR Cryptology ePrint Archive, pp. 46, 2018.

**Base64**  
O esquema de codificação base64 é usado para codificar dados binários em caracteres imprimíveis.

Podemos fazer um experimento rápido usando a linha de comando do OpenSSL:



O comando acima gerou uma sequência aleatória de 16 bits e depois, usando a opção -base64, converteu isso em uma string de texto base64.  
O base64 converte de 8 bits para uma representação ASCII de 6 bits. Isso é útil para **armazenamento e transmissão**, especialmente em casos onde o **manuseio de dados binários poderia levar a incompatibilidades entre sistemas**.  
Esse mecanismo é uma forma flexível de representar dados binários como ASCII, o que pode ser **armazenado e transmitido facilmente e universalmente**.

**base58**  
O esquema base58 foi introduzido pela primeira vez com o **Bitcoin** e é usado para **codificar inteiros em strings alfanuméricas**.  
A ideia principal por trás desse esquema de codificação é evitar **caracteres não alfanuméricos** e também aqueles que se parecem entre si e podem levar a **ambiguidade** — por exemplo, a letra “l” minúscula pode parecer o número “1”.  
Essa característica é especialmente útil porque **endereços Bitcoin não devem conter ambiguidade na representação dos caracteres**, caso contrário, isso poderia levar ao envio incorreto de bitcoins para um endereço inexistente ou incorreto — o que seria claramente uma perda financeira.  
Esse esquema de codificação engenhoso evita esse tipo de situação ao **ignorar caracteres semelhantes**.

Exploraremos **base58** e seu papel na geração de endereços Bitcoin em detalhes no **Capítulo 6: Arquitetura do Bitcoin**.

**Funções aleatórias verificáveis (Verifiable random functions – VRFs)**  
Uma **função aleatória verificável (VRF)** é uma **função de hash com chave** que utiliza criptografia de chave pública em vez de criptografia de chave simétrica, como em **MACs**.  
Ela é uma **variação de chave pública** de uma função de hash com chave.  
Nesse esquema, o hash é computado pelo detentor da **chave privada**, o qual é **publicamente verificável**, usando a **chave pública** para verificar se o hash está correto.

Uma VRF é um **gerador de número aleatório**. Ela recebe uma entrada arbitrária e produz uma **saída aleatória e uma prova**.  
A prova é usada para **verificar a correção** da saída. Mais formalmente, ela compreende três algoritmos: KeyGen, Evaluate e Verify.

* KeyGen recebe uma entrada aleatória e gera uma chave de verificação e uma chave secreta.
* Evaluate recebe a chave privada, uma mensagem, e produz uma **saída aleatória e uma prova**. Essa saída é **única e pseudoaleatória**.
* Por fim, o algoritmo Verify recebe a chave de verificação, a mensagem, a saída e a prova e verifica se a saída produzida pelo algoritmo de avaliação é de fato correta.

As VRFs são usadas em blockchains para **seleção de proponente de bloco** em protocolos de consenso de prova de participação (**proof-of-stake**).  
Alguns exemplos de tais protocolos incluem **Algorand**, **Cardano Ouroboros** e **Polkadot BABE**.

A VRF foi introduzida por **Micali et al.** em:  
Micali, Silvio, Michael Rabin, e Salil Vadhan. “Verifiable random functions.”  
In *Proceedings of the 40th Annual Symposium on the Foundations of Computer Science (FOCS ‘99)*, pp. 120–130.  
New York: IEEE Computer Society Press, 1999.  
O artigo original está disponível aqui: <http://nrs.harvard.edu/urn-3:HUL.InstRepos:5028196>

**Resumo**  
Este capítulo começou com uma introdução a alguns conceitos matemáticos básicos e à criptografia de chave assimétrica.  
Discutimos diversos construtos como **RSA** e **ECC**.  
Também realizamos alguns experimentos usando o OpenSSL para ver como os conceitos teóricos podem ser implementados na prática.  
Além disso, exploramos conceitos avançados e modernos como **provas de conhecimento zero** e construções relacionadas, juntamente com **diferentes tipos de assinaturas digitais**.

**No próximo capítulo**, exploraremos o **cativante mundo do consenso distribuído**, que é central para a integridade de qualquer blockchain e é uma área muito ativa de pesquisa.