CP 02 – Tradução Capítulo 10 – livro “The hardware hacking handbook”

Separando as diferenças: Análise de poder diferencial

Usando medidas de energia para aprender sobre o fluxo do programa tem claras implicações de segurança, mas e se nós podemos ir mais além do que apenas aprendendo sobre o fluxo do programa? É fácil imaginar um algoritmo onde o código tem o mesmo fluxo de programa independentemente dos dados sendo processados, mas com uma técnica poderosa chamada análise de energia diferencial (DPA), nós podemos aprender sobre os dados sendo processados por um dispositivo, mesmo se o fluxo do programa for exatamente o mesmo. No capítulo anterior, você aprendeu que a análise simples de energia usa a assinatura de energia de um dispositivo para determinar amplamente a operação que está realizando. Essas operações podem ser os loops em uma verificação de PIN ou as operações modulares em um cálculo RSA. Na análise de energia simples, podemos tratar cada traço individualmente. Por exemplo, em um ataque de análise de energia simples no RSA, podemos usar a ordem das operações modulares para recuperar uma chave. Na análise de energia diferencial, analisamos as diferenças entre conjuntos de traços. Usamos estatísticas para analisar pequenas variações em nossos traços, o que pode nos permitir determinar que dados o dispositivo está processando até mesmo até os bits individuais. Já que os bits individuais afetam apenas um punhado de transistores, você pode imaginar que o efeito no consumo de energia é mínimo. Na verdade, geralmente não é possível medir um único bit em um traço de energia (a menos que cause grandes diferenças operacionais, como na implementação didática do RSA). O que podemos fazer, no entanto, é capturar muitos milhares, milhões, bilhões de traços de energia e usar o poder das estatísticas (trocadilho intencional) para detectar um pequeno viés na corrente causado por um bit. O objetivo de um ataque de análise de energia diferencial é usar medidas de energia para determinar algum estado secreto e constante - tipicamente uma chave criptográfica - de um algoritmo que está processando dados no dispositivo alvo. Esta técnica incrivelmente poderosa foi publicada pela primeira vez em 1998 por Paul Kocher, Joshua Jaffe e Benjamin Jun no artigo adequadamente intitulado "Análise de Energia Diferencial". DPA é um algoritmo específico de análise de energia de canal lateral, mas o termo é usado genericamente para descrever todos os algoritmos relacionados no campo. Nós o usaremos como um termo genérico aqui também, a menos que seja especificado de outra forma. Antes de poder realizar um ataque de DPA, você precisa ser capaz de comunicar-se com o alvo e fazê-lo executar a operação criptográfica desejada. Você coletará medidas no alvo e registrará seu consumo de energia. Em seguida, você processará as medidas e executará o ataque na esperança de recuperar a chave de criptografia. Embora este ataque pareça semelhante ao ataque SPA descrito no Capítulo 9, a etapa de processamento difere substancialmente. Mas antes de nos aprofundarmos no processamento implementado em um ataque de DPA, você precisará entender qual efeito específico estamos explorando. Vamos começar olhando para um microcontrolador humilde; esses dispositivos digitais programáveis são quase garantidos de estar em qualquer produto hackeável. Dentro do Microcontrolador Se você olhasse profundamente dentro do seu microcontrolador, veria todas as linhas condutoras levando sinais de um lado do chip para o outro, como mostrado na Figura 10-1. Várias linhas de dados fluem de uma seção do chip para outra. Um microcontrolador de 8 bits tipicamente tem um único barramento de dados principal de 8 bits de largura. Essas linhas transportam dados, e alguns desses dados serão nosso alvo. Todas essas linhas eventualmente correm para um dos blocos de construção dos circuitos digitais, ou seja, transistores. Estes são transistores de efeito de campo (FETs), mas tudo o que nos importa é que eles são basicamente um interruptor; eles têm uma entrada que liga ou desliga a saída. Para alternar os FETs no final das linhas de barramento de dados, devemos mover essa linha de barramento de dados para cima ou para baixo. A entrada para o FET junto com todas as linhas intermediárias podem ser pensadas como um capacitor muito pequeno, e mover essa linha para cima ou para baixo realmente significa mudar a tensão através desse capacitor, o que significa que os valores de dados afetam diretamente as cargas nas capacitâncias internas Usando medidas de energia para aprender sobre o fluxo do programa tem claras implicações de segurança, mas e se nós podemos ir mais além do que apenas aprendendo sobre o fluxo do programa? É fácil imaginar um algoritmo onde o código tem o mesmo fluxo de programa independentemente dos dados sendo processados, mas com uma técnica poderosa chamada análise de energia diferencial (DPA), nós podemos aprender sobre os dados sendo processados por um dispositivo, mesmo se o fluxo do programa for exatamente o mesmo. No capítulo anterior, você aprendeu que a análise simples de energia usa a assinatura de energia de um dispositivo para determinar amplamente a operação que está realizando. Essas operações podem ser os loops em uma verificação de PIN ou as operações modulares em um cálculo RSA. Na análise de energia simples, podemos tratar cada traço individualmente. Por exemplo, em um ataque de análise de energia simples no RSA, podemos usar a ordem das operações modulares para recuperar uma chave. Na análise de energia diferencial, analisamos as diferenças entre conjuntos de traços. Usamos estatísticas para analisar pequenas variações em nossos traços, o que pode nos permitir determinar que dados o dispositivo está processando até mesmo até os bits individuais. Já que os bits individuais afetam apenas um punhado de transistores, você pode imaginar que o efeito no consumo de energia é mínimo. Na verdade, geralmente não é possível medir um único bit em um traço de energia (a menos que cause grandes diferenças operacionais, como na implementação didática do RSA). O que podemos fazer, no entanto, é capturar muitos milhares, milhões, bilhões de traços de energia e usar o poder das estatísticas (trocadilho intencional) para detectar um pequeno viés na corrente causado por um bit. O objetivo de um ataque de análise de energia diferencial é usar medidas de energia para determinar algum estado secreto e constante - tipicamente uma chave criptográfica - de um algoritmo que está processando dados no dispositivo alvo. Esta técnica incrivelmente poderosa foi publicada pela primeira vez em 1998 por Paul Kocher, Joshua Jaffe e Benjamin Jun no artigo adequadamente intitulado "Análise de Energia Diferencial". DPA é um algoritmo específico de análise de energia de canal lateral, mas o termo é usado genericamente para descrever todos os algoritmos relacionados no campo. Nós o usaremos como um termo genérico aqui também, a menos que seja especificado de outra forma. Antes de poder realizar um ataque de DPA, você precisa ser capaz de comunicar-se com o alvo e fazê-lo executar a operação criptográfica desejada. Você coletará medidas no alvo e registrará seu consumo de energia. Em seguida, você processará as medidas e executará o ataque na esperança de recuperar a chave de criptografia. Embora este ataque pareça semelhante ao ataque SPA descrito no Capítulo 9, a etapa de processamento difere substancialmente. Mas antes de nos aprofundarmos no processamento implementado em um ataque de DPA, você precisará entender qual efeito específico estamos explorando. Vamos começar olhando para um microcontrolador humilde; esses dispositivos digitais programáveis são quase garantidos de estar em qualquer produto hackeável. Dentro do Microcontrolador Se você olhasse profundamente dentro do seu microcontrolador, veria todas as linhas condutoras levando sinais de um lado do chip para o outro, como mostrado na Figura 10-1. Várias linhas de dados fluem de uma seção do chip para outra. Um microcontrolador de 8 bits tipicamente tem um único barramento de dados principal de 8 bits de largura. Essas linhas transportam dados, e alguns desses dados serão nosso alvo. Todas essas linhas eventualmente correm para um dos blocos de construção dos circuitos digitais, ou seja, transistores. Estes são transistores de efeito de campo (FETs), mas tudo o que nos importa é que eles são basicamente um interruptor; eles têm uma entrada que liga ou desliga a saída. Para alternar os FETs no final das linhas de barramento de dados, devemos mover essa linha de barramento de dados para cima ou para baixo. A entrada para o FET junto com todas as linhas intermediárias podem ser pensadas como um capacitor muito pequeno, e mover essa linha para cima ou para baixo realmente significa mudar a tensão através desse capacitor, o que significa que os valores de dados afetam diretamente as cargas nas capacitâncias internas

Alterando a Tensão em um Capacitor

Todos os tipos de capacitâncias dentro e ao redor do microcontrolador afetam o consumo de energia. Para efeitos da seguinte discussão, vamos nos referir a todas essas capacitâncias como um único capacitor. Se você estava prestando atenção na física do ensino médio, você pode lembrar que para aumentar a tensão através de um capacitor, você precisa aplicar uma carga, que tem que vir de algum lugar - mais usualmente através das linhas de energia. Um circuito integrado digital (CI) terá tanto linhas de energia VCC (positiva) quanto GND (terra). Se você estivesse monitorando o consumo de energia, você veria picos de corrente na linha VCC ao alternar de baixo para alto. Isso decorre das equações fundamentais em torno da mudança de tensão em um capacitor, que podem ser declaradas como "a corrente através de um capacitor está relacionada à capacitância C e à taxa de mudança de tensão", como mostrado aqui:

I = C dV/dt

Se tivermos uma tensão variável no capacitor (como o que acontece ao alternar de um estado baixo para alto), teremos uma corrente fluindo no circuito do qual o capacitor faz parte. Se a tensão estiver mudando em uma transição de baixo para alto, deveríamos ver um fluxo de corrente em uma direção. Se a tensão mudar em uma transição de alto para baixo, deveríamos ver esse fluxo de corrente invertido. Observar a magnitude e direção do fluxo de corrente nos permite inferir algo sobre as mudanças de tensão sobre o "capacitor", e assim sobre todo o circuito (incluindo as transições ocorrendo nos estados internos do barramento do microcontrolador).

Para ilustrar isso, vamos assumir que temos um microcontrolador que nos permite monitorar o consumo de corrente e o estado do barramento de dados interno. Se alterarmos duas linhas de dados enquanto monitoramos a corrente que entra no dispositivo, esperamos que os resultados dessa medição se pareçam com algo como a Figura 10-2. Quando os dados no barramento mudam, as linhas de dados mudam de estado simultaneamente em relação a um clock do sistema em pontos bem definidos no tempo. Nestes momentos, vemos picos de corrente resultantes das linhas de dados se alternando. Alternar as linhas de dados significa carregar e descarregar um capacitor, o que requer um fluxo de corrente.

A maioria dos barramentos de microcontroladores da vida real entram em um estado de pré-carregamento, que está no meio do caminho entre um um e um zero lógicos. Uma troca de estado lógico leva tempo, e o tempo depende da diferença de tensão a ser aplicada ao barramento (ou seja, a diferença de tensão entre o estado um e o estado zero). Através do pré-carregamento, essa diferença de tensão é constante e apenas metade da distância de uma troca completa de zero para um, não importa se há um zero ou um um no barramento. Isso resulta em operações de barramento levando menos tempo para atingir o estado final e toda a operação sendo mais confiável.

De Energia para Dados e de Volta

A maioria das medições que discutiremos neste livro tem como objetivo capturar a corrente do dispositivo em teste. A energia está relacionada à corrente com P = I × V; veja o Capítulo 2 para mais detalhes. Se o dispositivo tem uma tensão de operação constante, a energia e a corrente têm uma relação linear. Para o trabalho que segue, não precisamos de unidades específicas nessas medições, e um fator de escala linear (ou mesmo não linear) faz pouca diferença na aplicação dos resultados.

Por esse motivo, os termos corrente e energia são usados ​​indistintamente na seguinte discussão e para o restante deste livro. A nomenclatura comum para esses ataques é a de análise de energia, então você verá referência a invasores medindo a energia do dispositivo ou tendo traços de energia. Na maioria dos casos, isso não é preciso porque a corrente do dispositivo no circuito está sendo medida com ferramentas como sondas de corrente. (Para confundi-lo ainda mais, essas correntes são medidas por um osciloscópio em volts. Se você for especialmente pedante sobre a diferença entre energia e corrente, esteja avisado de que você pode achar a existência no campo da análise de energia totalmente impossível.)

Como invasores, podemos usar o mencionado estado de pré-carregamento para determinar diretamente o número de uns no número sendo manipulado. Este número é referido como o peso de Hamming (HW). O peso de Hamming de 0xA4 é 3 porque 0xA4 é 10010100 em binário e nele estão três uns. Com um barramento de 2 bits pré-carregado simples, nosso traço de consumo de energia se pareceria com a Figura 10-3.

Como resultado do pré-carregamento, o pico de energia depende apenas do número de uns no valor atual sendo enviado pelo barramento. Note que estamos considerando apenas o consumo de corrente do trilho VCC, razão pela qual não há picos negativos quando as linhas mudam para um estado baixo. Esse comportamento corresponde mais de perto ao que você veria em sistemas reais, já que você está observando a energia de apenas um trilho.

Na vida real, os microcontroladores geralmente vazam o peso de Hamming dos dados processados. Podemos confirmar isso ao calcular a média do consumo de energia em um momento em que sabemos quais dados estão sendo processados no barramento ao longo de muitas medições. A Figura 10-4 mostra um exemplo para um microcontrolador STM32F303.

Você pode se surpreender com o quão perfeitamente linear é este ajuste, mas nossas medições da vida real em microcontroladores muitas vezes realmente resultam em corresponder a este modelo. Medimos a queda de tensão sobre um resistor em série na linha VCC, então um aumento no consumo de energia (aumento do peso de Hamming) resulta em uma maior queda de tensão.

NOTA A palavra Hamming refere-se a Richard Hamming, um homem sensato que vivia por máximas como "Se a previsão de que um avião pode ficar no ar depende da diferença entre a Integração de Riemann e Lebesgue, eu não quero voar nele"Ele também desenvolveu a ideia da distância de Hamming em um artigo de 1950 intitulado "Códigos de Detecção e Correção de Erros". O objetivo principal do artigo era introduzir o código de Hamming, criando efetivamente a ideia de códigos de correção de erros. As ideias deste artigo são usadas em tudo, desde discos rígidos até comunicações sem fio de alta velocidade.

Exemplo Sexy XORy

Agora que podemos usar o consumo médio de energia para determinar a soma do número de bits que estão definidos como um em um dispositivo digital, vamos ver como poderíamos quebrar um dispositivo simples. Considere um circuito básico que faz um XOR de cada byte de entrada com uma chave secreta desconhecida, mas constante, de 8 bits. Em seguida, ele envia esses dados por uma tabela de pesquisa com valores conhecidos que substituem cada byte por outro valor, assim como um cifra de substituição, onde o byte de entrada original é substituído por um byte de saída correspondente em uma tabela de pesquisa, resultando em um resultado "criptografado".

Não temos acesso à saída deste dispositivo; tudo o que podemos fazer é enviar-lhe dados que ele faz um XOR e passa pela tabela de pesquisa. No entanto, podemos medir a energia deste dispositivo, como mostrado na Figura 10-5, inserindo um resistor shunt na linha VCC do dispositivo em teste.

Agora enviamos um monte de bytes de dados de entrada de 8 bits aleatórios para o dispositivo e registramos cada byte juntamente com o traço de energia. Acabamos com uma lista de dados enviados para o dispositivo, juntamente com o traço de energia associado medido durante essa operação, como mostrado na Figura 10-6.

Isso é tudo que precisamos para iniciar um ataque DPA, onde tentaremos recuperar a chave secreta.

Ataque de Análise de Energia Diferencial

Para o ataque DPA neste exemplo XOR da Figura 10-5, miramos um único bit da chave secreta de cada vez. Vamos descrever como quebrar o bit menos significativo (LSB), mas você pode estender isso para todos os 8 bits com um toque de criatividade.

Fundamental para esses ataques é a enumeração de chaves, que é uma forma sofisticada de dizer que fazemos suposições informadas sobre a chave. Tentamos todos os valores de chave possíveis, prevemos qual seria o consumo de energia se o dispositivo usasse esse valor de chave e comparamos nossas previsões com os traços de energia reais. O melhor ajuste é nosso candidato a chave.

Neste ponto, você está absolutamente certo em pensar: "Por que eu preciso de análise de energia em vez de simplesmente forçar bruto uma chave de 8 bits?" Para um ataque de força bruta, você precisa inserir uma chave e obter algum feedback do sistema sobre se a chave está correta. O problema aqui é que assumimos que a saída não está disponível, então você nunca poderia testar se a chave adivinhada estava correta. Com DPA, vamos obter algumas "dicas" sobre se uma chave adivinhada está correta. Na verdade, não aprendemos se a chave descriptografa os dados. O melhor teste de uma chave adivinhada seria tentar descriptografar alguns dados e ver se resulta em uma saída válida; se sim, fundamentalmente sabemos que a chave está correta. Com um ataque DPA, tecnicamente apenas ganhamos confiança em uma hipótese de chave ou suposição de chave. Se essa confiança for muito alta, podemos deduzir que a chave real é igual à nossa hipótese de chave sem precisar realizar uma descriptografia de teste. Mais crucialmente, posteriormente estenderemos este exemplo para chaves maiores que não podem ser forçadas. Por exemplo, aplicar DPA a uma chave de 128 bits é 128 vezes mais trabalhoso do que aplicá-lo a um único bit, pois podemos realizar ataques em bits de chave independentemente dos outros bits de chave. Compare isso com a força bruta, onde adivinhar uma chave de um único bit requer no máximo duas tentativas, mas adivinhar todos os 128 bits requer no máximo 2^128 tentativas. Isso é um número grande. É aproximadamente o número de formigas no universo se cada estrela no universo tivesse um bilhão de formigas rainhas, e cada formiga rainha tivesse uma colônia de um bilhão. Isso significa que com DPA, é viável quebrar uma chave de 128 bits, enquanto com força bruta, não é.

NOTA Você pode ter ouvido falar de computação quântica e sua capacidade de quebrar algoritmos criptográficos. Os mais afetados são os sistemas baseados em RSA e ECC, que são "trivialmente" quebrados usando ataques quânticos. No entanto, mesmo se considerarmos computadores quânticos, algoritmos simétricos como AES permanecem principalmente seguros. Atualmente, os melhores ataques quânticos conhecidos para algoritmos simétricos apenas reduzem pela metade o número efetivo de bits do algoritmo. Isso significa que quebrar a chave de 128 bits do AES-128 é tão difícil em um computador quântico quanto quebrar uma chave de 64 bits em um computador clássico, e o AES-256 sob ataques quânticos é tão forte quanto forçar bruto de 128 bits. Forçar bruto de uma chave de 64 bits é praticamente plausível para um estado-nação, e forçar bruto de uma chave de 128 bits é efetivamente impossível. Mas em comparação, um ataque DPA ao AES-256 é apenas cerca de duas vezes mais difícil do que um ataque DPA ao AES-128.

Prevendo o Consumo de Energia Usando uma Suposição de Vazamento

Para prever o consumo de energia do dispositivo, usaremos uma suposição de vazamento em combinação com nosso conhecimento do sistema. Supomos que o sistema vaza o peso de Hamming de todos os valores processados, mas temos um problema. Somos capazes de medir apenas o consumo de energia total e, portanto, o peso de Hamming total de todos os dados que estão sendo processados, em vez do peso de Hamming apenas do valor secreto no qual estamos interessados. Além disso, mesmo se conseguirmos isolar o valor secreto, muitos valores de 8 bits diferentes terão o mesmo peso de Hamming. Uma vez que este capítulo tem muitas mais páginas, você adivinhou corretamente que uma solução para essa dificuldade existe. Digamos que temos uma matriz de traços de energia chamada t[] e uma matriz chamada p[] de dados de entrada associados. Por exemplo, a entrada superior da Figura 10-6 teria p[0] = 0xAC. O traço de energia t[0] é uma matriz de valores de amostra, mostrada como o traço superior. Podemos aplicar o algoritmo DPA para gerar uma lista de diferenças para cada suposição de chave. A função simples apresentada na Listagem 10-1 simula o consumo de energia de um dispositivo alvo simples e adivinha um único bit por meio de um ataque DPA.

Primeiro, enumeramos todas as possibilidades para o byte sendo adivinhado. Para cada suposição possível do byte da chave, percorremos todos os traços de energia registrados. Usando os dados de entrada associados ao traço p[d] e à suposição i da chave secreta, podemos gerar uma saída hipotética h que é apenas igual ao que o microcontrolador teria calculado se tivéssemos adivinhado a chave corretamente. Finalmente, observamos o bit alvo (o LSB) na saída hipotética. Com base na suposição da chave, adicionamos cada traço de energia registrado t[d] a um dos dois grupos: aqueles em que pensamos que o LSB era um um, e aqueles em que pensamos que o LSB era um zero. Agora, considere a natureza dessa suposição. Se a suposição estiver incorreta, o que pensamos ter sido inserido na tabela de pesquisa não é o que realmente foi inserido no dispositivo, e, consequentemente, o que pensamos ter saído da tabela de pesquisa também não é o que realmente saiu. Agrupar pelo LSB incorreto significa que basicamente dividimos todos os traços de energia aleatoriamente em dois grupos. Nesse caso, você esperaria que o consumo médio de energia de cada grupo fosse mais ou menos o mesmo. Portanto, se você subtrair as médias umas das outras, você não deve obter nada além de talvez algum ruído. A Figura 10-7 mostra alguns exemplos dos dois grupos e a subtração resultante. Se nossa suposição estiver correta, o que pensamos estar sendo calculado é na realidade o mesmo que os dados que foram calculados no dispositivo. Portanto, movemos todos os traços de energia onde o LSB está realmente definido como 1 para um grupo e todos os traços onde o LSB está realmente definido como 0 para o outro grupo. Se esses uns e zeros consumirem uma quantidade ligeiramente diferente de energia, essa diferença deve ficar evidente se agruparmos grandes grupos de traços. Esperaríamos ver uma pequena diferença entre os grupos um e zero quando este bit é manipulado, como mostrado na Figura 10-8. Essa diferença (na Listagem 10-1) nos dá a parte diferencial da análise de energia diferencial. O poder dessa análise está em separar os traços da tabela mostrada na Figura 10-6 em dois grupos, o que nos permite fazer a média de muitos traços para reduzir o ruído, sem eliminar a contribuição do bit de interesse. Podemos ver o último pico na amostra 35 na Figura 10-8, o que demonstra que podemos ver a pequena contribuição do nosso LSB. Fazer a diferença entre esses dois grupos médios será chamado de tomar a diferença das médias (DoM). Mas esse pequeno pico de consumo de energia não seria perdido no ruído de tantas outras linhas sendo comutadas em chips da vida real? Bem, todo esse outro ruído é efetivamente distribuído uniformemente pelos dois grupos. A única diferença que permanece estatisticamente significativa entre os grupos é o LSB, o único bit que escolhemos para dividir nossos grupos. Quando fazemos a média de um número suficiente de tais traços, as contribuições de quaisquer outros bits que mudam cancelam-se. Um Ataque de DPA em Python

Como prova de conceito, o caderno Jupyter associado a este capítulo (https://nostarch.com/hardwarehacking/) implementa um ataque de DPA em nosso exemplo em Python. A função measure\_power(), parcialmente mostrada na Listagem 10-2, realiza um XOR dos dados de entrada usando um byte secreto e os passa por uma tabela de pesquisa.

```python

def measure\_power(din):

#byte secreto

skey = 0b10010111 # 0x97

#Calcular resultado

res = lookup[din ^ skey]

```

Listagem 10-2: Tabela de pesquisa que faz XOR dos dados de entrada com alguma chave secreta

Nos exemplos seguintes, a tabela de pesquisa é gerada aleatoriamente (ou seja, a matriz de pesquisa da Listagem 10-2). A tabela de pesquisa deve ser pelo menos uma bijeção, e se estivéssemos implementando um algoritmo de criptografia real, haveria mais considerações. No entanto, para o propósito desta demonstração, uma sequência permutada aleatoriamente funcionará igualmente bem. O uso de tal tabela de pesquisa demonstrará que não há um "problema" fundamental com o AES ou outro algoritmo que torne o ataque possível.

NOTA Os nomes das funções e variáveis mencionados neste texto referem-se ao código Python como parte do caderno Jupyter associado. Você deverá ser capaz de acompanhar sem ele, mas poderá executar o exemplo interativamente com o caderno.

Em vez de simplesmente executar a "função de criptografia", simularemos o consumo de energia de um pedaço de hardware executando essa função, o que facilitará o acompanhamento em um computador. Você verá mais tarde como realizar as medições em um pedaço de hardware real.

Simulando uma única medição de energia

Para simular uma única medição de energia, geraremos uma matriz com ruído de fundo aleatório para refletir a realidade das medições e sistemas ruidosos na função measure\_power(). Em seguida, inseriremos um pico de energia com base no número de uns no valor intermediário. Isso simula as medições de consumo de energia do sistema mostradas na Figura 10-5.

A Medição em Lote

Em seguida, realizamos a medição em lote. A função gen\_traces() chama a função measure\_power() com um número de entradas aleatórias enquanto registra o traço de energia resultante. Você pode especificar quantas medições realizar. (Veremos o efeito disso na taxa de sucesso do ataque mais tarde.) A Figura 10-9 mostra um único traço que "medimos", como plotado a partir do Python.

Enumerando as Possibilidades e Dividindo os Traços

Neste ponto, temos as matrizes de medição e de dados de entrada mencionadas anteriormente na seção "Previsão do Consumo de Energia Usando uma Suposição de Vazamento". Tudo o que precisamos fazer agora é enumerar as suposições de chave e dividir os traços de energia registrados em dois grupos com base no valor intermediário hipotético.

Na função dom(), supomos o valor intermediário com lookup[guess ^ p] e então verificamos o valor para ver se um bit específico está definido com a expressão (XX >> bitnum) & 1. Com base no valor desse bit, os traços são divididos em dois grupos. Em nosso exemplo, antes de estarmos usando o LSB, isso corresponderia a bitnum sendo definido como 0.

O Array de Diferença

Finalmente, subtraímos a média de cada grupo para obter o array de diferença. Como essa diferença se parece? Se a divisão foi feita corretamente, esperaríamos um grande pico em algum ponto. Olhe de volta para a diferença de médias nas Figuras 10-7 e 10-8. Você deve ver o pico positivo óbvio quando a separação de traços é feita corretamente, e assim sabemos que nossa suposição de chave está correta. O gráfico na Figura 10-8 é o resultado de uma suposição correta, onde dividimos os traços com base na suposição de que o byte da chave secreta era 0x97. O gráfico na Figura 10-7 mostra uma suposição de chave incorreta, onde dividimos os traços com base na suposição de que o byte da chave secreta é 0xAB. Conforme separamos os traços, mesmo em ambientes com ruído muito alto, eventualmente tudo o que não é o sinal DPA se dissipará, como você pode ver comparando as diferenças de médias da esquerda para a direita na Figura 10-10.

A Figura 10-10 mostra 100.000 traços usados à direita em oposição a 1.000 traços à esquerda. O resultado é que o ruído aleatório é ainda mais suprimido, e o sinal se torna ainda mais pronunciado.

Um Ataque Completo

Em seguida, determinamos o valor mais provável da chave de criptografia para cada bit calculando a diferença das médias para cada suposição de um bit específico. A partir de todas essas diferenças, encontramos o pico mais forte, que indica qual é a melhor suposição para a chave para esse bit. Ao executar o código, produzimos esta saída:

Melhor suposição para o bit 0: 0x97

Melhor suposição para o bit 1: 0x97

Melhor suposição para o bit 2: 0x97

Melhor suposição para o bit 3: 0x97

Melhor suposição para o bit 4: 0x97

Melhor suposição para o bit 5: 0x97

Melhor suposição para o bit 6: 0x97

Melhor suposição para o bit 7: 0x97

Determinamos o valor correto da chave de criptografia para cada bit.

Embora a DPA lide com bits individuais de cada vez, o uso daquela engraçada tabela de consulta em nossa função de criptografia de exemplo significava que conseguimos quebrar todos os oito bits da chave de criptografia adivinhando apenas um único bit. Este método funcionou porque um único bit da saída da tabela de consulta pode estar relacionado a todos os bits da entrada para a tabela. Esta entrada é a chave desconhecida de 8 bits combinada com os dados de entrada do algoritmo de 8 bits conhecidos.

O uso da tabela de consulta garante que, se nossa suposição do valor da chave estiver errada, a divisão dos traços em categorias de um e zero seria basicamente aleatória. Especificamente, a tabela de consulta é muito provavelmente não linear porque a randomizamos.

Se estivéssemos atacando apenas uma simples entrada de chave XOR sem a tabela de consulta, cada bit da chave estaria relacionado apenas a um bit do estado intermediário, o que significa que só teríamos sido capazes de determinar um bit da chave por bit de estado intermediário.

Conhecendo o Inimigo: Um Curso Intensivo sobre o Padrão de Criptografia Avançado

Quebrar nosso algoritmo fictício que funciona em um único byte não é tão emocionante, então agora vamos aplicar a DPA ao padrão de criptografia avançado (AES). O AES sempre opera em blocos de 16 bytes, o que significa que você deve criptografar 16 bytes de cada vez. O AES tem três possibilidades de comprimento de chave: 128 bits (16 bytes), 192 bits (24 bytes) ou 256 bits (32 bytes). Chaves mais longas geralmente significam criptografia mais forte, já que qualquer tipo de ataque de força bruta leva exponencialmente mais tempo para quebrar chaves mais longas.

Lidamos principalmente com o AES-128 aqui (embora você também possa aplicar facilmente ataques de canal lateral ao AES-192 ou AES-256) usando o modo Livro de Códigos Eletrônicos (ECB). No modo ECB, um bloco de 16 bytes de texto simples não criptografado executado através do AES-128-ECB com a mesma chave secreta sempre mapeia para o mesmo texto cifrado. A maioria das criptografias do mundo real não usa diretamente o modo ECB, mas sim usa vários modos de operação, como o encadeamento de blocos de cifra (CBC) e o Modo de Contador de Galois (GCM). Uma DPA direta no AES se aplicaria diretamente ao AES no modo ECB. E uma vez que você saiba como lidar com o AES no modo ECB, também pode estendê-lo para ataques ao AES CBC e AES GCM.

N O T A O AES foi especificado pelo Instituto Nacional de Padrões e Tecnologia dos EUA em 2001. Você verá referências ao AES como o cifrador Rijndael porque o padrão na verdade foi o resultado de uma competição, e Rijndael foi uma das entradas. Rijndael foi criado pelos criptógrafos belgas Joan Daemen e Vincent Rijmen, então na próxima vez que estiver desfrutando de uma cerveja belga, certifique-se de dedicar um brinde a eles. Para mais detalhes sobre o algoritmo AES-128, consulte "Criptografia Séria" (No Starch Press, 2018), de Jean-Philippe Aumasson, ou "Compreendendo Criptografia" (Springer, 2010), de Christof Paar e Jan Pelzl, e seu site complementar.

Na Figura 10-11, a chave secreta de 16 bytes é dada como R0Kk, onde k é o número do byte da chave. O primeiro subscrito indica a qual rodada essa chave se aplica; o AES usa uma chave de rodada de 16 bytes diferente para cada rodada. O texto simples de entrada é inserido, novamente com um subscrito indicando o número do byte. Cada byte da chave de rodada é XOR'd com cada byte do texto simples em uma operação referida como AddRoundKey. Note que para o AES-128, a primeira chave de rodada é a mesma que a chave AES; todas as outras chaves de rodada são derivadas da chave AES através do algoritmo de agendamento de chave. Para DPA no AES-128, precisamos extrair apenas uma chave de rodada da qual podemos derivar a chave AES.

Uma vez que a chave de rodada e o texto simples foram XOR

'd juntos na operação AddRoundKey, cada byte é passado por uma caixa de substituição (S-box), em uma operação referida como SubBytes. A S-box é uma tabela de consulta de 8 bits com um mapeamento um para um (ou seja, cada entrada mapeia para uma saída única). Isso também significa que é inversível; dada a saída da S-box, você pode determinar a entrada. A S-box é projetada para ter várias propriedades preferidas que podem desencorajar a criptoanálise linear e diferencial. (A definição exata dessas tabelas de consulta é irrelevante; só queremos observar que a S-box é mais do que apenas qualquer tabela de consulta antiga.)

As duas camadas seguintes distribuem ainda mais a entrada por vários bits de saída. A primeira camada é uma função chamada ShiftRows, que embaralha os bytes. Em seguida, a operação MixColumns combina 4 bytes de entrada para criar 4 bytes de saída, o que implica que, se um único byte mudar na entrada para MixColumns, todos os 4 bytes de saída serão afetados.

A saída de MixColumns se torna a entrada para a próxima rodada. Esta rodada possui uma chave de rodada, que será XOR'd com o texto da rodada de entrada novamente usando a operação AddRoundKey. As operações anteriores (SubBytes, ShiftRows e MixColumns) então se repetem. A consequência é que se invertermos um único bit no início do AES, ao final das 10 rodadas, deveríamos (em média) ver metade dos bits de saída se inverterem.

Todas as rodadas, exceto a última, terão exatamente as mesmas operações; apenas os dados que entram na rodada e a chave de rodada diferirão. A última rodada terá outra operação AddRoundKey em vez de uma operação MixColumns. No entanto, precisaremos atacar apenas a primeira rodada com DPA para extrair uma chave completa, então não estamos muito preocupados com essa última rodada!

Atacando o AES-128 Usando DPA

Para quebrar uma implementação do AES-128 com DPA, primeiro precisamos simular uma implementação do AES-128. O exemplo de XOR que estamos usando é basicamente os dois primeiros passos do AES: uma adição de chave (XOR) e uma consulta S-box. Para construir um verdadeiro ataque DPA ao AES, modificaremos o código de amostra do caderno Jupyter companheiro (se você ainda não o fez, agora é um bom momento para fazê-lo funcionar). Simplesmente precisamos alterar nossa tabela de consulta randomizada para ser a S-box AES adequada. Neste caso, estamos atacando a saída da S-box. O efeito não linear da S-box tornará mais fácil extrair a chave de criptografia completa.

Se você executar o código de amostra, ele deve produzir a saída na Figura 10-12, que mostra um traço para cada uma das três valores da variável de suposição: 0x96, 0x97 e 0x98. Estes são os traços de diferença para três dos 256 valores da variável de suposição. Quando a variável de suposição corresponde ao valor correto do byte da chave, você pode ver um grande pico.

Embora estejamos atacando apenas um único byte da criptografia AES-128, podemos repetir o ataque para cada byte de entrada para determinar a chave inteira de 16 bytes. Lembra como nos saímos adivinhando apenas sobre 8 bits? Não fizemos nenhuma suposição especial sobre qual dos 8 bits da chave quebramos. Portanto, podemos fazer o mesmo ataque em qualquer um dos bytes da chave.

Agora afirmamos que podemos quebrar todos os bytes da chave AES atacando 16 vezes e adivinhando apenas 8 bits para cada ataque! Isso é computacionalmente totalmente viável, enquanto fazer um ataque de força bruta de 2128 está fora de questão.

A força fundamental do DPA é que, em vez de forçar todo o espaço de chaves, dividimos o algoritmo criptográfico em subchaves e, em seguida, forçamos essas subchaves usando informações adicionais dos traços de energia para validar as suposições de subchave. Dessa forma, transformamos a quebra de implementações do AES-128 de algo impossível para uma realidade alcançável.

Ataque de Análise de Correlação de Potência

O ataque DPA assume que, para um dispositivo específico, você obterá uma diferença no consumo de energia quando um bit é 1 ou 0. Como explicamos, podemos usar qualquer um dos 8 bits extraídos da tabela de consulta para prever a chave. Essa redundância é algo que realmente podemos usar para fortalecer nosso ataque. Uma maneira direta seria usar cada bit como um "voto" separado no que seria o subconjunto de chave provável, mas podemos ser mais inteligentes. Podemos usar um ataque mais avançado chamado análise de correlação de potência (CPA), que modelará simultaneamente qualquer número de bits e, portanto, pode produzir um ataque mais forte. Em termos de DPA/CPA, isso significa que precisamos de menos traços para recuperar a chave. CPA foi introduzida por Eric Brier, Christophe Clavier e Francis Olivier no artigo CHES 2004 "Análise de Correlação de Potência com um Modelo de Vazamento". Apresentaremos a notação matemática juntamente com a implementação em Python para que você possa corresponder a teoria ao código do mundo real.

Até você realmente implementar o ataque, os detalhes vão escapar (acredite em nós), então pegue um papel e uma caneta e vamos mergulhar.

No DPA, basicamente estamos dizendo: "se algum bit intermediário varia, o consumo de energia varia com ele". Embora isso seja verdade, não captura toda a extensão da relação entre dados e consumo de energia. Consulte a Figura 10-4. Quanto maior o peso de Hamming de uma palavra (ou seja, mais bits definidos), maior o consumo de energia. É próximo de uma relação linear perfe

ita. Essa relação parece se manter para qualquer tipo de CMOS, então se aplica muito bem aos microcontroladores. Agora, como exploramos essa linearidade?

A ideia básica no DPA é fazer suposições de chave e prever o que um bit em um valor intermediário seria. No CPA, fazemos as mesmas suposições de chave, mas prevemos toda a palavra de um valor intermediário. Em nosso exemplo AES, prevemos a saída de 8 bits da S-box:

sbox[guess ^ input\_data[d]]

Agora, aqui vem a magia: após a previsão, calculamos o peso de Hamming desse valor previsto. Sabemos que está muito próxima de uma relação linear com o consumo real de energia. Portanto, se nossa suposição estiver correta, deveríamos ser capazes de encontrar uma relação linear entre o peso de Hamming das saídas da S-box e o consumo real de energia medido de um dispositivo. Se nossa suposição estiver incorreta, não veremos uma relação linear porque o peso de Hamming que calculamos para o valor previsto na verdade era o peso de Hamming para algum outro valor ainda desconhecido, e não para o valor que previmos. O que será muito útil para nós é encontrar a suposição que fornece essa relação linear.

Como explorar essa relação linear ficará evidente à medida que voltamos nossa atenção para um certo Sr. Pearson.

Coeficiente de Correlação

O coeficiente de correlação de Pearson r faz o que estamos procurando. Ele mede a relação linear entre amostras de duas variáveis aleatórias - em nosso caso, os traços de energia medidos e o peso de Hamming da saída da S-box para uma determinada suposição de chave. Por definição, o coeficiente de correlação de Pearson é +1 se essas forem perfeitamente relacionadas linearmente; ou seja, quanto maior o consumo de energia, maior o peso de Hamming. Se o coeficiente de correlação for -1, elas são perfeitamente negativamente correlacionadas; ou seja, um peso de Hamming mais alto se correlaciona com um consumo de energia menor.

Uma correlação negativa pode ocorrer na prática por várias razões, então estamos tipicamente interessados no valor absoluto do coeficiente de correlação. Se a correlação for 0, não há relação linear alguma, e para nossos propósitos práticos, isso significa que para uma determinada suposição de chave, os traços medidos não correspondem significativamente ao peso de Hamming da S-box. Por essa observação, podemos testar o quão boa é uma suposição e comparar diferentes suposições simplesmente olhando para o valor absoluto da correlação de Pearson. A suposição com a correlação absoluta mais alta vence e, portanto, provavelmente é a chave real!

Primeiro, Alguma Nomenclatura

Estamos prestes a introduzir um monte de variáveis em equações que mapeiam para expressões Python no caderno. Para sua conveniência, fornecemos o mapeamento na Tabela 10-1.

Converter de equação para Python é uma parte importante do processo seguinte, juntamente com muitos dos ataques que você lerá no futuro. Criar tabelas de mapeamento simples como a Tabela 10-1 pode facilitar muito sua vida. Se você tiver o código do companheiro funcionando, mantenha esta página aberta para converter rapidamente entre equação e código.

Calculando os Dados para Correlacionar

Para calcular o coeficiente de correlação, precisaremos de uma tabela de medições de energia real de um dispositivo (veja a Tabela 10-2) e uma coluna de medições de energia hipotéticas (veja a Tabela 10-3). Vamos primeiro olhar para a Tabela 10-2, a medição de energia, que é gerada usando o código no caderno companheiro.

O número do traço d representa uma operação de criptografia específica, texto simples e o traço de energia correspondente. Para toda a operação, registraríamos T amostras do traço de energia, cada amostra sendo uma medição de energia em um momento diferente durante a operação. O número total de amostras em cada traço depende da taxa de amostragem de nossa medição e de quanto tempo a operação dura. Por exemplo, se nossa operação AES levasse 10ms (0,01s), e nosso osciloscópio registrasse 100 milhões de amostras por segundo (MS/s), teríamos 0,01 × 100.000.000 = 1.000.000 de amostras (ou seja, T = 1.000.000). Em cenários práticos, T pode ser praticamente qualquer coisa, mas muitas vezes está na faixa de 100 a 1.000.000 de amostras. Nosso ataque CPA considerará cada amostra independentemente, então tecnicamente precisamos apenas de uma única amostra para cada traço (mas essa única amostra precisaria estar no momento certo).

Para as medições de energia hipotéticas, não temos mais um eixo de amostra (ou tempo). Em vez disso, consideramos qual seria o consumo de energia hipotético para o mesmo número de traço (o mesmo índice d), dado um palpite de chave i. O que aconteceu com o tempo então? Anteriormente, dissemos que o ataque poderia ter sucesso com um único ponto de amostra no "momento certo". O "momento certo" na verdade significa o tempo em que o dispositivo está realizando a operação na qual modelamos nosso consumo de energia hipotético. Isso significa que nossa medição hipotética não precisa de um índice de tempo porque estamos definindo o tempo como sendo durante a operação de interesse. Com a medição física, não sabemos quando essa operação ocorreu, então precisamos registrar um traço de energia mais longo que inclua essa operação (mas também inclui outras coisas que nosso ataque eliminará). A Tabela 10-3 mostra a tabela de valores hipotéticos com a qual estamos trabalhando neste exemplo.

Para cada palpite de chave, calculamos o peso de Hamming da saída da S-box e colocamos os resultados em uma tabela, com uma coluna para cada palpite, numerada de 0 a 255. Nossa hipótese é que se o byte de chave secreta for 0x00, as medições de energia se parecerão com a coluna 0; se o byte de chave secreta for 0x01, as medições de energia se parecerão com a coluna 1; e se o byte de chave secreta for 0xFF, as medições de energia serão como na coluna 255. Queremos ver qual coluna (se houver) se correlaciona fortemente com as medições de energia física.

Anteriormente, usamos tabelas de traços de energia medidos. Aqui, representaremos essas tabelas pela notação td,j, onde j = 0,1,..., T – 1 é o índice de tempo no traço, e d = 0,1,..., D – 1 é o número do traço. Se você estiver acompanhando o exemplo de código no caderno Jupyter para esta seção, estaremos indexando uma variável chamada traces[d][j]. Como mencionamos antes, se o atacante souber exatamente onde ocorreu uma operação criptográfica, ele precisaria medir apenas um único ponto, de modo que T = 1. Para cada número de traço d, o atacante também sabe o texto simples correspondente a esse traço de energia, definido como pd. A variável pd é equivalente a input\_data[d] no código do companheiro, e é a primeira coluna nas Tabelas 10-2 e 10-3.

Para o tempo correto (j = 35) e palpite de chave (i = 0x97), a correlação é significativamente maior. Claro, a "tabela completa" teria todos os pontos de amostra (tempos), com o índice j variando de 0 a T - 1, juntamente com todos os palpites de chave de 0 a I - 1. O ponto final dos palpites de chave I - 1 neste exemplo é 0xFF, pois nosso modelo de vazamento foi baseado em uma entrada de um único byte, que só pode assumir os valores de 0x00 a 0xFF. Mostramos alguns exemplos de alguns pontos de amostra para manter esta tabela apresentável.

Atacando AES-128 Usando CPA

Agora que podemos usar CPA para detectar vazamentos, vamos percorrer um exemplo de ataque a um único byte do algoritmo AES-128 como fizemos na seção "Atacando AES-128 Usando DPA" na página 310. Usaremos a função measure\_power() novamente, com o objetivo de atacar esse único byte. Estenderemos os exemplos anteriores para criar uma função intermediate(), que representa o valor hd,i = l(w(pd, i)). Para um determinado byte de entrada de texto simples e um palpite da chave, esta função retorna o peso de Hamming esperado do valor intermediário. O ataque CPA usará isso ao comparar o vazamento esperado com o vazamento medido real.

Loop de Soma

Observe na equação do coeficiente de correlação de Pearson que há efetivamente três somas sobre todos os traços. Para esta implementação inicial, calcularemos algumas dessas somas e as quebraremos neste formato:

Em Python, primeiro calculamos todas as médias usando o palpite de chave atual. Então, para cada traço, atualizamos todas as variáveis de soma. Uma soma é gerada para cada ponto de amostra apresentado na entrada. Novamente, o resultado do coeficiente de correlação de Pearson (que é usado pelo ataque CPA) determina onde ocorreu a operação específica sensível; você não precisa saber antecipadamente quando a criptografia ocorreu.

Cálculo e Análise da Correlação

Para finalizar o ataque, geramos o traço de correlação combinando as somas. Plotamos o traço de correlação para diferentes números de palpites com a expectativa de que o pico mais alto ocorra com o palpite de chave correto (veja a Figura 10-13).

Os traços de correlação devem mostrar uma forte correlação no ponto em que o palpite corresponde ao valor secreto usado pelo dispositivo. O pico na Figura 10-13 e nos gráficos de correlação em geral mostram uma forte correlação positiva, mas você pode acabar com uma forte correlação negativa para o palpite de chave correto se medir o consumo de energia de forma contrária ao que o modelo prevê. Essa correlação negativa pode ser porque você está medindo no caminho de GND em vez do caminho de VCC, ou sua sonda pode estar conectada em polaridade inversa, ou sua configuração de medição pode causar leituras invertidas por algum outro motivo. Portanto, para determinar o palpite de chave correto, apenas olhamos para o valor absoluto do pico de correlação.

O ataque CPA é uma forma de quebrar implementações criptográficas que geralmente são muito seguras para um ataque DPA porque o CPA considera o vazamento de todos os 8 bits (para um sistema de 8 bits). O ataque DPA considera apenas um único bit. O princípio de um ataque CPA é baseado na observação de que você pode relacionar linearmente o peso de Hamming de uma variável intermediária ao consumo de energia de um dispositivo e que emprega a correlação para explorar esse relacionamento.

Experimente ajustar o número de traços para baixo em ambos os ataques DPA e CPA até que eles falhem em recuperar a chave correta de forma confiável. Você provavelmente descobrirá que por volta de 200 traços, o ataque DPA falhará em recuperar a chave correta, enquanto o ataque CPA recuperará a chave correta até cerca de 40 traços. Ambos os sistemas simulados têm a mesma quantidade de ruído; o ataque CPA usa a contribuição de vários bits para obter resultados muito melhores.

Modelos de Vazamento e Valores Sensíveis

Um modelo de vazamento descreve como os valores de dados processados em um dispositivo são expressos em um canal lateral. Até agora, usamos o modelo de vazamento de peso de Hamming, onde o consumo de energia tinha alguma relação linear com o número de bits definidos em uma linha de I/O. Como valor sensível, escolhemos um estado intermediário logo após um valor secreto ser misturado com nossos dados de entrada conhecidos e após uma operação não linear. O vazamento de peso de Hamming ocorreu devido ao fenômeno de pré-carregamento do barramento. No entanto, nem todo vazamento no chip é devido a barramentos pré-carregados. Outro modelo de vazamento comumente encontrado é a distância de Hamming (HD). O modelo HD é baseado no fato de que quando um registro passa de um estado para o próximo, o consumo de energia depende apenas do número de bits que mudam de estado. Portanto, ao usar este modelo, você se preocupará apenas com a diferença de número de bits entre dois ciclos de clock. A Figura 10-14 mostra um exemplo do HD para um registro.

Essa progressão mostra que o vazamento reflete as mudanças no estado do registrador. Se esse registrador estiver segurando a saída de uma S-box, você precisaria saber (ou adivinhar) o estado anterior desse registrador para quebrar o estado atual. Implementações criptográficas em hardware, como um periférico AES em um microcontrolador onde o algoritmo não está sendo executado como um processo de software, são muito mais propensas a serem vulneráveis ao vazamento de distância de Hamming. Como geralmente têm apenas um pequeno número de interconexões entre registradores (em comparação com o barramento de dados principal), elas não levam as linhas de dados a um estado de pré-carga, o que nos leva a detectar uma distância de Hamming em oposição a um peso de Hamming. Ao atacar esses dispositivos, precisamos calcular o consumo de energia hipotético de uma mudança, o que significa que precisamos determinar o estado anterior de tal registrador sensível. Pode ser que o estado anterior tenha sido apenas o último byte de entrada usado, ou poderia ter sido a saída da última vez que a operação de criptografia foi executada.

Determinar o valor anterior em circuitos especializados para implementar AES-128 pode ser esperado apresentar mais desafios porque esse valor agora dependerá de detalhes de projeto de hardware (como mostrado anteriormente na Figura 10-11). Os designers de hardware têm mais flexibilidade do que os designers de software, e ao implementar AES-128, eles podem optar por usar 16 cópias das tabelas de pesquisa S-box sendo executadas em paralelo ou compartilhar uma única tabela de pesquisa S-box entre todos os bytes de entrada, realizando sucessivamente a pesquisa, como mostrado na Figura 10-15. Pode ser necessário algum trabalho investigativo para identificar qual abordagem eles escolheram.

A escolha da implementação dependerá do propósito do dispositivo: um microcontrolador de propósito geral provavelmente aceitará um throughput mais lento quando um núcleo AES muito pequeno e de baixa potência estiver sendo projetado, enquanto um núcleo AES projetado para operar em um disco rígido ou controlador de rede compensará quaisquer restrições de potência ou tamanho do dispositivo que possam existir para acomodar um throughput multi-Gbps. Você pode deduzir algo sobre a estrutura medindo o número de ciclos de clock que o AES leva e depois dividindo pelo número de rodadas. Com aproximadamente 1 clock por rodada, todas as S-boxes (e outras operações AES dentro da rodada) são executadas em paralelo. Com aproximadamente 4 clocks por rodada, operações como SubBytes e MixColumns são executadas em ciclos de clock separados. Quando você chega a mais de 20 clocks por rodada, SubBytes provavelmente é implementado com uma única S-box.

Quanto menos você souber sobre um alvo, mais precisará usar tentativa e erro para determinar como ele implementa a criptografia. Se você descobrir que a saída da S-box de um dispositivo não está vazando, tente adivinhar bytes após a operação MixColumns (descrita anteriormente na seção "Conheça o Seu Inimigo: Um Curso Intensivo sobre Padrão de Criptografia Avançado"). Se o método de peso de Hamming não mostrar correlação, tente a abordagem de distância de Hamming. O artigo "Análise de Canal Lateral do Motor Criptográfico AVR XMEGA" de Ilya Kizhvatov fornece um ótimo exemplo disso em circuitos práticos, mostrando como quebrar o periférico AES XMEGA. Você também encontrará um tutorial passo a passo repetindo esse ataque XMEGA como parte do projeto ChipWhisperer, onde você pode experimentar esses resultados por si mesmo.

DPA em Hardware Real (mas Ainda Brinquedo)

O Capítulo 8 explicou como realizar medições de energia para SPA. A configuração de aquisição neste capítulo para DPA é a mesma, então vamos construir a partir daqui. Não tente atacar um dispositivo real até entender como o DPA funciona e ter simulado o ataque Python. Acredite nos especialistas: verifique triplo cada passo que você der. É fácil para um único bug em sua aquisição ou análise impedir que você veja qualquer vazamento.

Vamos inserir o AES em uma estrutura de software simples, com o firmware realizando uma operação de criptografia. Você pode usar qualquer biblioteca AES para a criptografia, como a biblioteca avr-crypto-lib de código aberto. Você até encontrará portas dessa biblioteca para Arduino (https://github.com/DavyLandman/AESLib/, por exemplo). O Exemplo 10-3 mostra um exemplo de código-fonte capaz de receber dados pela porta serial e iniciar uma criptografia.

Este exemplo apresenta um protocolo serial muito simples; você envia 16 bytes da chave em ASCII, 16 bytes de texto simples, e o sistema responde com os dados criptografados.

Por exemplo, você poderia abrir uma porta serial e enviar o seguinte texto: . Teste sua implementação pesquisando "Vetores de Teste AES-128" na internet.

Comunicando-se com um Dispositivo-Alvo

Ao definir seu próprio protocolo serial para enviar e receber dados, a comunicação com o alvo deve ser simples. Como nos exemplos de SPA, enviaremos alguns dados para o alvo e registraremos seu consumo de energia durante a operação AES. Se você acompanhou o caderno complementar, ele mostrou como realizar a medição em um dispositivo virtual; simplesmente substitua a função de medição por uma chamada ao dispositivo físico.

Os exemplos de medição simulada anteriores realizaram este ataque em um único byte, mas você precisará enviar 16 bytes para o dispositivo real. Você pode optar por realizar o ataque em qualquer byte arbitrário ou iterar por cada byte.

Novamente, dispare na borda de subida da linha de E/S para determinar os pontos de dados exatos de interesse. Ao mirar na primeira rodada do AES, por exemplo, mova o código trigger\_high() mostrado na Listagem 10-3 dentro da função AES de forma que a linha esteja alta apenas por volta do tempo da sua operação sensível (como a saída da pesquisa na S-box).

Velocidade de Captura do Osciloscópio

Assim como no ataque SPA, você pode determinar experimentalmente a taxa de amostragem necessária para qualquer plataforma ou dispositivo. Em geral, o ataque DPA requer taxas de amostragem consideravelmente mais altas do que o SPA, porque classificaremos os dados em um dos muitos grupos com base em pequenas variações na energia. Em contraste, o ataque SPA geralmente corresponde apenas a grandes variações na aparência das trilhas de energia, com o resultado de que o SPA pode operar em condições com ruído e jitter de tempo muito maiores do que o DPA pode.

Em geral, ao atacar uma implementação de software como AES em um microcontrolador, deve ser suficiente amostrar o dispositivo cerca de 1 a 5 vezes a velocidade do clock. Atacar implementações de hardware requer uma taxa de amostragem mais alta, frequentemente (com trocadilho) de 5 a 10 vezes a velocidade do clock. Essas são, no entanto, regras vagas no máximo; sua escolha de taxa de amostragem dependerá do vazamento do seu dispositivo, configuração de medição e qualidade do osciloscópio. Certos métodos de amostragem, como a amostragem síncrona usada na plataforma ChipWhisperer, também podem relaxar esses requisitos para que você possa amostrar até mesmo na velocidade do clock em si (1 vez a velocidade do clock) e ter um ataque bem-sucedido.