Capítulo 6d: Oblivious Linear Evaluation

Neste capítulo vamos procurar apresentar algumas construções essenciais às provas de conhecimento zero necessárias à construção de esquemas de autenticação.

Links
+Capítulo 6: Provas de Conhecimento e Assinatura Digital
+Capítulo 6a: Segurança de Esquemas de Assinatura Digital
+Capítulo 6b: Circuitos Algébricos e Aritméticos.
+Capítulo 6c: Computação Cooperativa
+Capítulo 6e: "Garbled Circuits"

Oblivious Transfer (de novo)

Funcionalidade $\binom{n}{\kappa}$
Vimos no +Capítulo 2b: Cifras Assimétricas uma construção do protocolo "Oblivious Transfer" genérico da forma $\binom{n}{\kappa}$-OT onde intervêm dois agentes, o **sender** e o **receiver.** Recordemos a funcionalidade básica:

- O agente **sender** disponibiliza n mensagens $m_1, m_2, \cdots, m_n$ para serem transferidas
- O gente **receiver** escolhe um conjunto I de κ das mensagens e recebe condidencialmente do 1º agente, κ todos os m_i com $i \in I$; no final da transferência
  - O agente **sender** não identifica as κ mensagens transferidas mas tem a certeza que não mais do que κ das mensagens m_i foram transmitidas,
  - O agente **receiver** não conhece qualquer m_i se $i \notin I$.

Como vimos na referência acima, este protocolo genérico é implementado usando técnicas assimétricas o que normalmente exige representações e computações usando estruturas algébricas complexas. Quando integrado em outras técnicas, que iremos referir em seguida, estas implementações transformam-se na componente computacionalmente mais complexa do protocolo que muito provavelmente será incomportável em termos de recursos.

Por isso é frequente usar formas particulares do protocolo com implementações que são muito mais eficientes das que são referidas no capítulo 2b. Para isso destacamos alguns casos particulares, nomeadamente $\binom{2}{1}$ e o caso mais geral $\binom{n}{n-1}$, que têm um tipo de implementação que pode ser muito eficiente: a "Learning Parity with Noise"

Funcionalidade $\binom{2}{1}$ OT

Para uma sessão identificada com **sid**

Choose(sid, b)

i. O **sender** envia um "obliviousness criterion" (OC) para o **receiver;**
ii. O **receiver** escolhe um bit b, regista na sua memória esse bit e manda as chaves públicas apropridas para o **sender.**
iii. Se o OC é satisfeito o **sender** aceita a informação recebida como representantes da escolha do bit b; regista-as na sua memória para ser usada em futuras transferências. Se o OC não é satisfeito, então aborta todo o protocolo.

Transfer(sid, m_0, m_1)

i. se a memória do **sender** não contém nenhum registo de escolha termina o protocolo em falha e bloqueia quaisquer funcionalidades futuras.
ii. Se existir tal registo com a informação pública que dispõe (chaves públicas) envia m_b para o **receiver** .

"Learning Parity with Noise" (LPN)

Baseado nos artigos *Statistically "Sender-Private OT from LPN and Derandomization"* e *"Zero-Knowledge Proofs from Learning Parity with Noise"* acessíveis a partir daqui.

Representa-se por $\mathcal{B}(\epsilon)$, sendo ϵ um racional positivo, o gerador pseudo-aleatório de bits, dito *gerador de Bernoulli* , visto como elementos $b \in \mathbb{F}_2$, tal que

$$\mathbb{P}[\, b = 1 \,|\, b \leftarrow \mathcal{B}(\epsilon)\,] \;=\; \epsilon$$

O gerador $\mathcal{B} \equiv \mathbb{F}_2$ é o gerador sem qualquer desvio que pode ser implementado por um XOF a partir uma qualquer "seed" apropriada. Para um parâmetro de segurança λ representamos por $\mathcal{B}^\lambda$ o gerador de vetores $a \in \mathbb{F}_2^\lambda$, e construído como λ cópias concorrentes de $\mathcal{B}$.

A forma mais direta de implementar um gerador de Bernoulli $\mathcal{B}(\epsilon)$ com a precisão de n bits, é o algoritmo.

$$\mathcal{B}(\epsilon) \equiv \vartheta\, w \leftarrow \{0,1\}^n \,.\, \textsf{if } \sum_{i=1}^n w_i\, 2^{-i} \leq \varepsilon \textsf{ then } 1 \textsf{ else } 0$$

Aqui $\hat{w} \equiv \sum_{i=1}^n 2^{-i}\, w_i$ é o designado *racional de Lebesgue* determinado pela string de bits w. Em muitos CPU's, $\hat{w}$ pode ser calculado em tempo constante; por isso, este é um processo usual para gerar uniformemente racionais no intervalo $[0\,,\,1]$.

O gerador $\mathsf{LPN}_{\lambda,\epsilon}(\mathsf{s})$, para um segredo $\mathsf{s} \leftarrow \mathcal{B}^\lambda$, usa parâmetros λ e ϵ definindo-se do seguinte modo

$$\mathsf{LPN}_{\lambda,\epsilon}(\mathsf{s}) \equiv \quad \vartheta\, a \leftarrow \mathcal{B}^\lambda \,.\, e \leftarrow \mathcal{B}(\varepsilon) \,.\, \vartheta\, t \leftarrow \mathsf{s}\cdot a + e \,.\, \langle\, a, t\,\rangle$$

Aqui $\mathsf{s}\cdot a \equiv \sum_i \mathsf{s}_i \times a_i$ denota o produto interno dos dois vetores.

> *Para $\mathsf{s} \neq 0$ e $\varepsilon > 0$ o gerador $\mathsf{LPN}_{\lambda,\varepsilon}(\mathsf{s})$ é indistinguível do gerador pseudo-aleatório $\mathcal{B}^{\lambda+1}$.*
>
> *Adicionalmente, é uma crença, que escolhendo parâmetros λ, ϵ apropriados, não existe nenhum algoritmo $\mathcal{A}$, probabilístico e polinomial no modelo quântico que, usando $\mathsf{LPN}_{\lambda,\epsilon}(\mathsf{s})$ como oráculo, consiga determinar o segredo s.*

$\binom{2}{1}$-OT em LPN

Numa implementação de $\binom{2}{1}$-OT o oráculo LPN vai ser implementado por um XOF que recebe como argumento uma "seed" α e um comprimento ℓ.
O output dessa função é uma sequência de pares $\{\langle\, a_i, u_i\,\rangle\}_{i=1}^\ell$ em que $a_i \in \mathbb{F}_2^\lambda$ e $u_i \in \mathbb{F}_2$.

Pode-se empacotar a sequência de pares $\{\langle\, a_i, u_i\,\rangle\}_{i=1}^\ell$ numa matriz $\mathsf{A} \equiv \{a_i\}_{i=1}^\ell \in \mathbb{F}_2^{\ell\times\lambda}$, cujas linhas são os vetores a_i, e num vector $\mathsf{u} \equiv \{u_i\}_{i=1}^\ell \in \mathbb{F}_2^\ell$ cujas componentes são os bits u_i.

No que se segue $\bar{b}$ denota a negação do bit b; isto é $\bar{b} + b = 1$.

Choose(b)

a. O **sender** escolhe o par (α, ℓ) e a função XOF e envia essa informação para o **receiver;** esta informação determina completamente a sequência $\{\langle a_i, u_i \rangle\}_{i=1}^{\ell}$ que passa a formar o "oblivious criterion" ; ambos os agentes podem construir estes elementos.

b. o **receive**r gera o segredo $\mathsf{s} \leftarrow \mathcal{B}^{\lambda}$ e os diversos bits de "ruído" $\{e_i \leftarrow \mathcal{B}(\epsilon)\}_{i=1}^{\ell}$, os "tags" LPN $\{t_i \leftarrow a_i \cdot \mathsf{s} + e_i\}_{i=1}^{\ell}$. Regista esta informação na sua memória.
Calcula o par de chaves públicas
$$\{\, \langle\, p_i^0 \leftarrow t_i + b \cdot u_i \;\;,\;\; p_i^1 \leftarrow t_i + \bar{b} \cdot u_i \,\rangle \,\}_{i=1}^{\ell}$$
e envia-as para o **sender**.

c. o **sender** recolhe as chaves públicas e verifica a igualdade $p_i^0 + p_i^1 = u_i$.
Se, para algum $i \in \{1, \cdots, \ell\}$ a igualdade não se verifica então termina em falha.
Se se verificar a igualdade, para todo i, então regista este par de chaves na sua memória para transferência futura.

Transfer(m_0, m_1)

a. O **sender** conhece as mensagens $m_0, m_1 \in \mathbb{F}_2$. Para as cifrar gera aleatoriamente uma sequência de bits $\{\, r_i \leftarrow \mathcal{B} \,\}_{i=0}^{\ell}$ com um peso de Hamming (número de bits 1) limitado a uma parâmetro δ, e calcula
$$a \leftarrow \textstyle\sum_i r_i\, a_i \quad , \quad c_0 \leftarrow m_0 + \sum_i r_i\, p_i^0 \qquad c_1 \leftarrow m_1 + \sum_i r_i\, p_i^1$$
O criptograma é o triplo $\langle\, a\,,\, c_0\,,\, c_1 \,\rangle$ que é enviado ao **receiver**.

b. O **receiver** conhece o segredo s. Por isso pode calcular $a \cdot \mathsf{s}$.
Temos $a \cdot \mathsf{s} = \sum_i r_i (a_i \cdot \mathsf{s}) = \sum_i r_i\, (e_i + t_i) =$

$$= (\textstyle\sum_i r_i e_i) + \left\{ \begin{array}{rcl} \sum_i r_i \cdot (p_i^0 + b \cdot u_i) & = & c_0 + m_0 + b \cdot (\sum_i r_i u_i) \\ \sum_i r_i \cdot (p_i^1 + \bar{b} \cdot u_i) & = & c_1 + m_1 + \bar{b} \cdot (\sum_i r_i u_i) \end{array} \right.$$

Para simplificar a notação e a interpretação sejam
$$\mathsf{error} \equiv \textstyle\sum_i r_i e_i \quad , \quad \mathsf{off} \equiv \sum_i r_i u_i$$
Ambos valores são desconhecidos do **receiver** porque não conhece os r_i (apesar de conhecer tanto os e_i como os u_i). No entanto conhece duas equações
$$m_0 = c_0 + (a \cdot \mathsf{s}) + \mathsf{error} + b \cdot \mathsf{off}$$
$$m_1 = c_1 + (a \cdot \mathsf{s}) + \mathsf{error} + \bar{b} \cdot \mathsf{off}$$
Portanto se $b = 0$ o **receiver** tem, a menos de error, $m_0 \simeq c_0 + (a \cdot \mathsf{s})$.
Igualmente se for $b = 1$ o **receiver** usa a 2ª equação para obter $m_1 \simeq c_1 + (a \cdot \mathsf{s})$.

Resta discutir qual é o efeito de error. Os e_i têm uma grande probabilidade de serem 0. Os r_i têm um número limitados de valores 1. Por isso, com grande probabilidade os produtos $r_i e_i$ são nulos.
Pode acontecer, com baixa probabilidade, que seja $\sum_i r_i e_i = 1$.
Para corrigir esta possibilidade pode-se repetir a operação Transfer gerando um novo conjunto de valores $\{r_i\}$ e calculando de novo

$$m_b \leftarrow c_b + (a \cdot \mathsf{s})$$

Pode-se repetir este processo vária vezes, sempre com novos $\{r_i\}$; recolhe-se os várias resultados m_b e, no final, escolhe-se o valor que aparece mais vezes.

Paraa repetição, se à partida for definido que vão ser executadas t iterações nesta operação Transfer, então gera-se todos os r_i para todas as repetições de uma só vez. Equivalentemente cada r_i é gerado não como um bit simples mas sim como um vetor de t bits (um por iteração). Como as gerações são independentes, podem ser executadas em paralelo: a j-ésima iteração usa a sequência $\{r_{i,j}\}_{i=1}^{\ell}$. O criptograma também é calculado em paralelo e cada componente em (a, c_0, c_1) adquire uma dimensão extra correspondente ao índice $j = 1..t$ de repetição; i.e. a passa a ser uma matriz $t \times \lambda$ e c_0, c_1 passam a ser vetores de dimensão t. O resultado $\vec{m_b} \leftarrow c_b + (a \cdot \mathsf{s})$ passa a ser um vetor também de dimensão t. O valor de $v \in \{0,1\}$ que ocorre em maioria no vector $\vec{m_b}$ será o valor final de m_b.

Mensagens arbitrárias
Um factor limitativo deste protocolo resulta de as mensagens transferidas terem apenas um bit. No entanto, para mensagens m_0, m_1 de comprimento arbitrário, o protocolo continua a ser usável; basta aplicá-lo individualmente a cada posição de bit em m_0, m_1.

Por exemplo, se tivermos $m_0, m_1 \in \mathbb{F}_2^n$ (e.g se cada mensagem tiver n "bits") então

a. O número de amostras $\{\langle a_{i,j}, t_{i,j}\rangle\}_{i\in[\ell],j\in[n]}$ passará a ter $\ell \times n$ elementos. Cada segmento de ℓ pares é dedicado a uma posição na mensagem.
b. Do mesmo modo tem-se $\{p^0_{i,j}, p^1_{i,j}, e_{i,j}\}_{i\in[\ell],j\in[n]}$ onde cada segmento de ℓ elementos está associada a uma posição $j \in [n]$
c. Os u_i, r_i, c_0, c_1, b e $\overline{b}$ deixam de ser bits e passam a ser vetores de n bits. Nomeadamente b_j determina a escolha feita para a posição j.

d. Para contrariar o efeito do **error** usa-se um ℓ suficientemente grande e repete-se todo o protocolo $3, 5, 7, \cdots$ vezes, selecionando para cada posição j o valor do bit m_j que ocorre mais vezes desde que tenha uma maioria significativa.

Nesta opção, uma vez que as mensagens têm vários "bits", é possível interpretá-las como inteiros num determinado intervalo $[-\delta, \delta - 1]$. Para mensagens de n bits teremos $\delta \leq 2^{n-1}$.
Aqui é possível e desejável usar na geração de "noise" e, em alternativa ao gerador de Bernoulli $\mathcal{B}(\epsilon)$ apropriado apenas para bits, um gerador que produz inteiros segundo uma distribuição Gaussiana discreta de média 0 e um desvio padrão $\texttt{sigma} = \alpha\,\delta$.
O parâmetro α é um racional positivo, tipicamente $\alpha \sim 0.1..0.3$, que serve para ajustar o equilíbrio entre a segurança e a eficiência do algoritmo.
Pode ver detalhes na seguinte página da documentação SageMath.

$\binom{N}{N-1}$-OT em LPN

O protocolo $\binom{2}{1}$-OT implementado em LPN pode ser generalizado para um protocolo $\binom{N}{N-1}$-OT
modificando as operações Choose e Transfer.

No que se segue, para todo $n > 0$, representamos por $[n]$ o intervalo de inteiros $\{0..n-1\}$.

Choose(b)

Neste protocolo $b \in [N]$ denota o índice da mensagem que vai ser excluída das transferências legítimas; o criptograma c_b não pode ser decifrado corretamente pelo **receiver** porque este agente não conhece uma chave privada que o permita.

O **sender** escolhe o par (α, ℓ) e a função XOF e envia essa informação para o **receiver;** esta informação determina completamente a sequência $\{\langle\, a_i, u_i \,\rangle\}_{i=1}^{\ell}$ que passa a formar o "oblivious criterion" ; ambos os agentes podem construir estes elementos.

a. o **receiver** gera N segredos $\mathsf{s}_k \leftarrow \mathcal{B}^{\lambda}$, se $k \neq b$, e $\mathsf{s}_b \leftarrow \perp$. Para todo $k \in [N]$ e todo $i \in [\ell]$, calcula $t_{k,i}$ da seguinte forma

$$t_{i,k} \leftarrow \begin{cases} \vartheta\, e_{i,k} \leftarrow \mathcal{B}(\epsilon) \;\centerdot\; a_i \cdot \mathsf{s}_k + e_{i,k} & \text{se} \quad k \neq b \\ u_i - \sum_{j \neq b}\, t_{i,j} & \text{se} \quad k = b \end{cases}$$

Regista esta informação na sua memória.
Construímos, para cada $i \in [\ell]$, um vetor em $\mathcal{B}^N$

$$\mathbf{t}_i \equiv \{t_{i,k} \mid k \in [N]\}$$

e envia-os para o **sender** como chaves públicas.

c . o **sender** recolhe todas os vetores de chaves públicas $\mathbf{t}_i$ e verifca as igualdades

$$\sum_{k\in[N]} \mathbf{t}_{i,k} = u_i$$

Se, para algum $i \in [\ell]$ a igualdade não se verifica então termina em falha.
Se se verificar a igualdade então regista todos os $\mathbf{t}_i$ na sua memória para transferência futura.

Transfer$(m_0, m_1, \cdots, m_{N-1})$

a. O **sender** conhece as mensagens $m_k \in \mathbb{F}_2$, $k \in [N]$ e, para cada $i \in [\ell]$ as chaves públicas $\mathbf{t}_i$.
Para as cifrar gera aleatoriamente uma sequência de bits $\{ r_i \leftarrow \mathcal{B} \}_{i=0}^{\ell}$ com um peso de Hamming (número de bits 1) limitado a uma parâmetro δ , e calcula

$$a \leftarrow \sum_i r_i\, a_i \quad , \quad c_k \leftarrow m_k + \sum_i r_i \cdot \mathbf{t}_{i,k} \quad \text{para todos } k \in [N]$$

O criptograma é o tuplo $\langle a\,,\, c_0\,,\, \cdots\,,\, c_{N-1} \rangle$ que é enviado ao **receiver**.

b. O **receiver** conhece os segredos $\mathbf{s}_k$ para todo $k \in [N]$. Sabe que $\mathbf{s}_b = \perp$ e que para todo $k \neq b$ pode calcular $a \cdot \mathbf{s}_k$. Sabe também que se verifica, para todo $k \neq b$, a relação

$$m_k = c_k + (a \cdot \mathbf{s}_k) + \mathbf{error}_k$$

sendo $\mathbf{error}_k = \sum_i r_i \cdot e_{i,k}$ um valor desconhecido (porque o **receiver** não conhece os r_i) mas com elevada probabilidade de ser nulo.

Procedendo como no protocolo $\binom{2}{1}$-OT , pode-se reforçar esta probabilidade, iterando ambas as operações t vezes; para isso cifra-se usando vetores $r_i \in \mathcal{B}^t$. As iterações são independentes e podem ser executadas em paralelo. A **sender** produz t criptogramas distintos, um por iteraçao.
O **receiver** toma este conjunto de criptogramas e calcula, para cada um, um resultado

$$m_k \leftarrow c_k + (a \cdot s_k) \quad \text{para todo } k \neq b$$

Toma-se como resultado final de m_κ , para cada $k \neq b$, o valor em maioria nas diferentes iterações; assim obtém-se, com elevada probabilidade, o valor da mensagem inicial.

Finalmente para mensagens $\{m_k\}_{k\in[N]}$ de comprimento arbitrário, tal como no caso, $\binom{2}{1}$OT, usa-se o protocolo de mensagens binárias para cada posição nas mensagens.
