洛夫什·哈尔坎达尼 跟随

2019 年 6 月 16 日 · 13 分钟阅读 · 听

Sigma 协议的零知识证明

照片由Niklas Hamann在Unsplash上拍摄

Sigma 协议是 3 阶段协议，用于在不公开值本身的情况下证明某种关系中的值知识。例如。离散对数的知识，即给定 g 和，在不透露的情况下 y 证明 in 的知识这篇文章将

相等性。一些代码示例 `x g^x = y x`. 在 Python 中是链接的，它们在椭圆曲线上工作以显示各种结构。在最后一节中，Sigma 协议被描述为一个简单但引人入胜的泛化的特例，并且使用该泛化描述了一些协议。
这篇文章的目的不是解释零知识的理论概念或结果，而是为没有太多学术背景的人介绍零知识。

一些术语：被证明知识的价值被称为*见证*。关系的其他元素统称为*instance*。证明证人知识的实体称为*证明者*。证明者需要说服证人知识的另一方称为*验证者*。*证明者通过发送验证者验证的证明*来说服验证者。在离散日志示例中，`x` 见证人是只有证明者知道的，`g` 并且 `y` 是证明者和验证者都知道的实例。

Sigma 协议遵循以下 3 个步骤：

1. 承诺：证明者生成一个随机数，创建对该随机性的承诺并将承诺发送给验证者。
2. **Challenge**：验证者得到承诺后，生成一个随机数作为挑战，发送给证明者。重要的是验证者在获得承诺之前不要发送挑战，否则证明者可以作弊。
3. 响应：证明者接受挑战并使用在步骤 1 中选择的随机数、挑战和见证人创建响应。然后，证明者会将响应发送给验证者，验证者将进行一些计算，并且会或不会相信证人的知识。

上述协议是一个交互式协议，意味着证明者和验证者都需要在线并且能够发送消息来完成协议。但它可以通过Fiat-Shamir技巧使其成为非交互式的。这实质上意味着验证者不会在第 2 步中发送质询，但证明者将通过使用散列函数对*特定于该协议执行的某些*内容进行散列来模拟质询。一个可能的候选者*是*第 1 步中的承诺和实例数据。在这种非交互协议中，证明者可以创建证明，任何验证者都可以对其进行验证。

一个例子

让我们考虑离散对数示例的证明知识，以更好地理解 Sigma 协议。`x` 在 $g^x = y$ 不泄露的情况下证明知识 `x`，

1. 证明者生成一个随机数 `r`，创建一个承诺 $t = g^r$ 并发送给 `t` 验证者。
2. 验证者存储 `t`、生成随机挑战 `c` 并将其发送给证明者。

证明者总是生成一个新的随机值 r 并因此生成一个新的 是很重要的 $t = g^r$ 。如果证明者与验证者使用相同 r ，则验证者可以学习 x 。这就是

在协议

i) 的第一次执行中，证明者创建 r 并因此创建 r 的方式 $t = g^r$ 。

ii) 验证者发送挑战 c1 和

iii) 证明者发送 $s1 = r + x*c1$.

在协议的第二次执行中，

i) 证明者再次选择相同 r 并因此生成相同 $t = g^r$ 并将其发送给验证者。

ii) 现在验证者会知道证明者已经选择了相同的选项 r ，因为他之前已经看到了 t 。验证者发送另一个挑战 c2 。

iii) 证明者然后发送 $s2 = r + x*c2$

验证者因此有 2 个线性方程 $s1 = r + x*c1$ 和 $s2 = r + x*c2$ 。由于它相同 r ，验证者现在有 $s1- x*c1 = s2- x*c2$ 。既然它知道 c1 而且 c2 ， $x = (s2-s1)/(c2-c1)$.验证者已获悉 x 。

证明者无法预测挑战也很重要。如果他可以，那么他可以在不知道的 x 情况下证明知识 x 。让我们看看如何

1. 假设证明者预测挑战是 c 。然后他 t 以不同的方式构建。 t 现在等于 g^s*y^{-c} 。这 s 是第三步中发送的响应证明者。他 s 随机生成。
2. 验证者 c 按预期发送挑战。
3. 证明者现在将发送 s 他在第一步中选择的。

验证器将照常检查是否 g^s 等于 $y^c.t$ 。这将是真实的，因为 $y^c.t = y^c * g^s * y^{-c} = g^s * y^c * y^{-c} = g^s *1 = g^s$ 。

因此验证者确信证明者知道 x 没有证明者曾 x 在上述过程中使用过。

上述协议也称为 Schnorr 识别协议或 Schnorr 协议。

使协议非交互

现在，如果证明者和验证者使用 Sigma 协议的非交互版本，则可以避免与验证者的交互，但这需要证明者使用哈希函数，这样他就无法预测哈希函数的输出，否则他可以如上所述作弊. 非交互式版本的另一个优点是验证器实现非常简单，因为它不需要随机数生成器。因此，非交互式版本是:

1. 证明者生成一个随机数 r 并创建一个承诺 $t = g^r$ 。证明者哈希 g ， t 并 y 获得挑

验证者现在生成相同的质询 c， $Hash(g, y, t)$ 并再次检查是否 g^s 等于 $y^c.t$。演示此协议的 Python 代码。

现在在某些实现中，大小 t 远大于 c 或 s （ t 是组元素， c 并且 s 是字段元素）。在这样的设置中，证明者和验证者参与协议的变体，其中证明者不发送 (t, s) 但 (c, s). 这是如何：

1. 证明者生成一个随机数 r 并创建一个承诺 $t = g^r$。证明者哈希 g， t 并 y 获得挑战 c。 $c = Hash(g, y, t)$.
2. 证明者创建对挑战的响应为 $s = r + c*x$。证明者将元组发送 (c, s) 给验证者。

验证者现在将：

1. 使用方程重构 t $g^s = y^c.t$。因此 $t = g^s y^{-c}$。
2. c 使用重建挑战 $c = Hash(g, y, t)$。如果这个重构的挑战 c 等于证明者发送的挑战，则验证者确信证明者知道 x.

上述非交互协议是 Schnorr 签名的主要思想。您可以将 Schnorr 签名视为与公钥（ x in g^x ）相对应的密钥知识的证明，其中除了我们上面看到的其他内容之外，正在签名的消息只是在挑战中散列。

让我们考虑更多使用非交互式版本的 Sigma 协议示例。证明者和验证者都可以创建相同的挑战。

协议 1. 离散日志的平等

假设证明者知道相关的证人 x， $g^x = y$ 以及 $h^x = z$ 在哪里 g， h 和是实例。为了说服验证者，他们采用以下协议 y z

1. Prover 在承诺步骤中生成 2 个承诺， $t1 = g^r$ 并 $t2 = h^r$ 承诺相同的随机数 r。
2. 证明者创造挑战 $c = Hash(g, h, y, z, t1, t2)$。
3. 证明者创建响应 $s = r + c*x$。他现在发送 $(t1, t2, s)$ 给验证者。现在，即使承诺不同，两个承诺的响应也是相同的。

协议 2. 离散日志的连接 (AND)

有时，需要对多个关系进行证明，其中每个关系都可以使用 sigma 协议来证明。然后可以将许多 sigma 协议组合在一起。让我们考虑**AND**组合。为了证明证明者知道 a 和 b 使得 $g^a = P$ **AND** $h^b = Q$，证明者遵循以下协议：

1. Prover 生成 2 个随机数 `r1` 和 `r2` 和 对应的承诺 $t1 = g^{r1}$ 和 $t2 = h^{r2}$。
2. 证明者创造挑战 `c = Hash(g, h, P, Q, t1, t2)`。
3. 证明者创建响应并发 `s1 = r1 + a*c` 送给 `s2 = r2 + b*c` 验证 `(t1, t2, s1, s2)` 者。
4. 验证程序检查 g^{s1} equals $P^c * t1$ 和 h^{s2} equals $Q^c * t2$。

以上协议相当于并行执行2个“离散对数知识”协议。绑定多个执行的东西是挑战，这在两个执行中都是相同的。该协议可以简单地推广到超过 2 个离散日志。协议的示例代码。

协议 3. 离散日志的析取 (OR)

现在考虑一个**OR**组合。假设有 2 个关系 $g^a = P$ 和 $h^b = Q$。证明者要么知道要么， `a` 但 `b` 不能同时知道两者。为了让验证者相信他知道其中之一 `a` 或 `b` 不透露他知道哪一个，证明者使用上述两种“并行协议”，但在其中一种中作弊。验证者知道证明者作弊，但不知道是哪一个。假设证明者知道 a 所以他只能诚实地运行“一个协议”（$g^a = P$）。

1. 证明者生成一个随机数 `r1`。
2. Prover $t1 = g^{r1}$ 像往常一样创建他的第一个承诺。
3. 这里证明者作弊。他对关系使用了两种不同的挑战。此外，他不知道见证人的关系的响应是随机挑选的。因此，对于第二个协议，他选择 `c2` 随机挑战和 `s2` 随机响应。
4. 现在，他构造了与平常不同的第二个承诺 $t2 = h^{s2} * Q^{-c2}$。
5. 为了获得第一个协议的挑战 `c1`，证明者像往常一样散列以获得 `c = Hash(g, h, P, Q, t1, t2)`. 现在 `c1 = c- c2`。
6. 证明者创建响应 `c1 = r1 + a*c1` 证明者现在发送 `(t1, t2, c1, s1, c2, s2)`

示例代码。

协议 4. Pedersen 承诺开启的知识

Pedersen 承诺是一种承诺方案，它让提交者对消息进行承诺， m 因为随机性 $g^m h^r$ 在哪里 r 。它允许提交者创建多个具有相同值的承诺，通过每次选择不同的随机性来区分这些承诺。这对 (m,r) 被称为承诺的开放。为了证明承诺中的开放知识 $P = g^m * h^r$ ：

1. Prover 生成 2 个随机数 $r1$ ， $r2$ 但创建了一个单一的 t 承诺 $g^{r1} * h^{r2}$ 。
2. 证明者创造挑战 $c = Hash(g, h, P, t)$ 。
3. 证明者创建响应并发 $s1 = r1 + m*c$ 送给 $s2 = r2 + r*c$ 验证 $(t, s1, s2)$ 者。
4. 验证器检查是否 $g^{s1} * h^{s2}$ 等于 $t * P^c$

请注意，这与 2 个离散日志的结合不同，因为验证者不知道 g^m 或 h^r 单独知道它们的乘积。这可以再次简单地推广到向量 Pedersen 承诺，即 $g1^{m1} * g2^{m2} * g3^{m3} * .. * h^r$ 。示例代码。

议定书 5. 2 项 Pedersen 承诺的平等开放

为了证明 2 个 Pedersen 承诺的开放 P 和 Q 相等，即 $P = g_1^a * h_1^b$ 和 $Q = g_2^a * h_2^b$ ，遵循以下协议：

1. Prover 生成 2 个随机数 $r1$ 和 $r2$ 并创建 2 个承诺 $t1 = g_1^{r1} * h_1^{r2}$ 和 $t2 = g_2^{r1} * h_2^{r2}$ 。
2. Prover 提出挑战 $c = Hash(g_1, h_1, g_2, h_2, P, Q, t1, t2)$
3. 证明者创建响应并发 $s1 = r1 + a*c$ 送给 $s2 = r2 + b*c$ 验证 $(t1, t2, s1, s2)$ 者。
4. 验证器检查是否 $g_1^{s1} * h_1^{s2}$ 等于 $P^c * t1$ 和 $g_2^{s1} * h_2^{s2}$ 等于 $Q^c * t2$

这可以简单地推广到向量 Pedersen 承诺或超过 2 个 Pedersen 承诺。示例代码。

协议 6. 2 Pedersen 承诺中的信息平等

为了证明 2 个承诺被提交到相同的消息但具有不同的随机性，即消息在哪里，并且 $P = g_1^a * h_1^b$ 是随机性，遵循以下协议： $Q = g_2^a * h_2^d$ a b d

2. 证明者创造挑战 $c = Hash(g_1, h_1, g_2, h_2, P, Q, t1, t2)$

3. 证明者创建响应 $s1 = r1 + a*c$ ， $s2 = r2 + b*c$ 并发 $s3 = r3 + d*c$ 送给 $(t1, t2, s1, s2, s3)$ 验证者。

4. 验证程序检查 $g_1^{s1} * h_1^{s2}$ equals $P^c * t1$ 和 $g_2^{s1} * h_2^{s3}$ equals $Q^c * t2$ 。

示例代码。

请注意，每当我们证明某些见证值在不同的关系中是相同的时，相同的随机值将用于它们相应的承诺中。相同的随机值 $r1$ 用于上述协议中的 a 消息 $t1$ 。 $t2$ 在之前的开放相等协议中，相同的随机值 $r1$ 和 $r2$ 用于 $t1$ 和 $t2$ 。离散对数协议的相等性也是如此。同样在上述协议中，使用了不同的生成器 g_1 ， g_2 ， h_1 ， h_2 。但是如果生成器在两个承诺中相同，则协议仍然有效，即 $g_1 = g_2$ 和 $h_1 = h_2$ 。

协议 7. 离散对数的不等式

假设证明者知道离散登录 a 。他想说服验证者 $g^a = P$ a 不等于另一个离散登录 b 。 $h^b = Q$ 证明者可能不知道，但他可以通过比较和 b 来检查它 b 是否相同。验证者可能会也可能不会。以下协议最初在本文第 6 节中描述。 $a\ h^a\ Q\ b$

1. Prover 生成一个随机值 r 。

2. Prover 创建一个承诺 $C = h^{a*r} * Q^{-r}$ 。让我们调用 $a*r$ as $alpha$ 和 $-r$ as $beta$ 。

3. Prover 现在需要证明 when C is not 1 , $h^{alpha} * Q^{beta} = C$ **AND** $g^{alpha} * P^{beta} = 1$ 。

   $g^{alpha} * P^{beta} = 1$ 是真的，因为 $g^{alpha} * P^{beta} = g^{a*r} * P^{-r} = P^r * P^{-r} = 1$ 。

4. 现在 1 和 C 都可以被视为 Pedersen 对不同生成器的消息 $alpha$ 和随机性的承诺，，，，。我们已经在协议 5 中看到了这一点。证明者执行协议 5 来获得. 然后他发送给验证者。 $beta\ h\ Q\ g\ P\ (t1, t2, s1, s2)\ (C, t1, t2, s1, s2)$

5. 验证者检查 C 不等于 1，然后按照协议 5 运行他的验证程序 $(t1, t2, s1, s2)$ 。

示例代码显示了完整的协议。示例代码使用椭圆曲线，因此 1 等价于曲线上的单位元（无穷远点）。该代码将变量 $iden$ 用于通过从自身减去曲线点获得的变量。

态，那么将其视为一个函数，它为输出提供与其输入类似的结构。数之和的指数可以看作是同态 $g^{a+b} = g^a * g^b$ 。
因此，假设同态被定义为一个函数 f ，其中 $f(a) = g^a$ 。为了证明 的知识 a ，给定公共输入 $f(a)$ ，即 g^a ，执行以下协议：

1. 证明者生成一个随机数并发 r 送给 $f(r) = g^r$ 验证者。
2. 验证者发送一个挑战 c 。
3. 证明者发送响应 $s = r + c*a$ 。
4. 验证者计算 $f(s) = g^s$ 并检查它是否等于 $f(r) * f(a)^c = g^r * g^{a*c}$ 。

正如视频进一步描述的那样，上述概括可用于证明上述某些协议。让我们试试：

协议 8. 离散日志的平等

$g^x = y$ 并且 $h^x = z$ 可以看作是这个同态的输入和输出 $f(a) = (g^a, h^a)$ 。所以 $f(x) = (g^x, h^x)$ 。现在使用上面的概括：

1. 证明者生成一个随机数 r 并将其发送 $f(r) = (g^r, h^r)$ 给验证者。
2. 验证者发送一个挑战 c 。
3. 证明者发送响应 $s = r + c*a$ 。
4. 验证者计算 $f(s) = (g^s, h^s)$ 并检查它是否等于 $f(r) * f(a)^c$ 。

$$
\begin{aligned}
& f(r) * f(a)^c \\
&= (g^r, h^r) * (g^x, h^x)^{*c} \\
&= (g^{r+c*x}, h^{r+c*x}) \\
&= (g^s, h^s)
\end{aligned}
$$

协议 9. 离散日志的连接 (AND)

$g^a = P$ **AND** $h^b = Q$ 可以看作是同态 $f(a, b) = (g^a, h^b)$。现在使用上面的概括：

1. 证明者生成 2 个随机数并发 r1 送给 r2 验证 $f(r1, r2) = (g^{r1}, h^{r2})$ 者。
2. 验证者发送一个挑战 c 。

$= (g^{r1}, h^{r2}) * (g^a, h^b)^c$

$= (g^{r1+c*a}, h^{r2+c*b})$

$= (g^{s1}, h^{s2})$

协议 9. Pedersen 承诺开启的知识

同态可以看成 $f(m, r) = g^m * h^r$ 。同样，使用上面的概括：

1. 证明者生成 2 个随机数并发 r1 送给 r2 验证 $f(r1, r2) = g^{r1} * h^{r2}$ 者。
2. 验证者发送一个挑战 c 。
3. 证明者发送响应 (s1, s2) where $s1 = r1 + c*m$ 和 $s2 = r2 + c*r$ 。
4. 验证者计算 $f(s1, s2) = g^{s1} * h^{s2}$ 并检查它是否等于 $f(r1, r2) * f(m, r)^c$ 。

$f(r1, r2) * f(m, r)^c$

$= (g^{r1} * h^{r2}) * (g^m * h^r)^c$

$= (g^{r1} * h^{r2}) * (g^{m*c} * h^{r*c})$

$= g^{r1+m*c} * h^{r2+r*c}$

$= g^{s1} * h^{s2}$

我认为可以推广更多协议，但我现在很无聊；）