Lecture Notes 24

114 Bit Commitment Using Hash Functions

The analogy between bit commitment and hash functions described above suggests a bit commitment scheme based on hash functions, as shown in Figure 114.1.

	Alice		Bob
	To $\mathbf{commit}(b)$:		
1.		$\xleftarrow{r_1}$	Choose random string r_1.
2.	Choose random string r_2.		
	Compute $c = H(r_1 r_2 b)$.	$\xrightarrow{c}$	c is commitment.
	To $\mathbf{open}(c)$:		
3.	Send r_2.	$\xrightarrow{r_2}$	Find $b' \in \{0,1\}$ such that $c = H(r_1 r_2 b')$. If no such b', then fail. Otherwise, b' is revealed bit.

Figure 114.1: Bit commitment using hash function.

The purpose of r_2 is to protect Alice's secret bit b. To find b before Alice opens the commitment, Bob would have to find r_2' and b' such that $H(r_1 r_2' b') = c$. This is akin to the problem of inverting H and is likely to be hard, although the one-way property for H is not strong enough to imply this. On the one hand, if Bob succeeds in finding such r_2' and b', he has indeed inverted H, but he does so only with the help of r_1—information that is not generally available when attempting to invert H.

The purpose of r_1 is to strengthen the protection that Bob gets from the hash properties of H. Even without r_1, the strong collision-free property of H would imply that Alice cannot find c, r_2, and r_2' such that $H(r_2 0) = c = H(r_2' 1)$. But by using r_1, Alice would have to find a new colliding pair for each run of the protocol. This protects Bob by preventing Alice from exploiting a few colliding pairs for H that she might happen to discover.

115 Bit Commitment Using Pseudorandom Sequence Generators

There are many ways to use a pseudorandom sequence generator G for bit commitment. One such way is shown in Figure 115.1. Here, ρ is a security parameter that controls the probability that a cheating Alice can fool Bob. We let $G_\rho(s)$ denote the first ρ bits of $G(s)$.

Assuming G is cryptographically strong, then c will look random to Bob, regardless of the value of b, so he will be unable to get any information about b.

The purpose of r is to protect Bob against a cheating Alice. Alice can cheat if she can find a triple (c, s_0, s_1) such that s_0 opens c to reveal 0 and s_1 opens c to reveal 1. Such a triple must satisfy

	Alice		Bob
	To **commit**(b):		
1.		$\xleftarrow{r}$	Choose random string $r \in \{0,1\}^\rho$.
2.	Choose random seed s. Let $y = G_\rho(s)$. If $b = 0$ let $c = y$. If $b = 1$ let $c = y \oplus r$.	$\xrightarrow{c}$	c is commitment.
	To **open**(c):		
3.	Send s.	$\xrightarrow{s}$	Let $y = G_\rho(s)$. If $c = y$ then reveal 0. If $c = y \oplus r$ then reveal 1. Otherwise, fail.

Figure 115.1: Bit commitment using PRSG.

the following pair of equations:

$$\left.\begin{array}{rcl} c & = & G_\rho(s_0) \\ c & = & G_\rho(s_1) \oplus r. \end{array}\right\} \tag{1}$$

It is sufficient for her to solve the equation

$$r = G_\rho(s_0) \oplus G_\rho(s_1) \tag{2}$$

for s_0 and s_1 and then choose $c = G_\rho(s_0)$.

One might ask why Bob needs to choose r? Why can't Alice choose r, or why can't r be fixed to some constant? If Alice chooses r, then she can easily solve (2) and cheat. If r is fixed to a constant, then if Alice ever finds a triple (c, s_0, s_1) satisfying (1), she can fool Bob every time. While finding such a pair would be difficult if G_ρ were a truly random function, any specific PRSG might have special properties, at least for a few seeds, that would make this possible. For example, suppose $r = 1^\rho$ and $G_\rho(\neg s_0) = \neg G_\rho(s_0)$ for some s_0. Then (2) could be solved by taking $s_1 = \neg s_0$. By having Bob choose r at random, r will be different each time (with very high probability), and a successful cheating Alice would be forced to solve (1) in general, not just for one special case.

116 Bit Commitment Schemes

The three bit commitment protocols of the previous section all have the same form. We abstract from these protocols a cryptographic primitive, called a *bit commitment scheme*, which consists of a pair of *key spaces* $\mathcal{K}_\mathcal{A}$ and $\mathcal{K}_\mathcal{B}$, a *blob space* $\mathcal{B}$, a *commitment* function

$$\mathbf{enclose} : \mathcal{K}_\mathcal{A} \times \mathcal{K}_\mathcal{B} \times \{0,1\} \to \mathcal{B},$$

and an *opening* function

$$\mathbf{reveal} : \mathcal{K}_\mathcal{A} \times \mathcal{K}_\mathcal{B} \times \mathcal{B} \to \{0,1,\phi\},$$

where ϕ means "failure". We say that a blob $c \in \mathcal{B}$ *contains* $b \in \{0,1\}$ if $\mathbf{reveal}(k_A, k_B, c) = b$ for some $k_A \in \mathcal{K}_A$ and $k_B \in \mathcal{K}_B$.

These functions have three properties:

	Alice		Bob
	To $\mathbf{commit}(b)$:		
1.		$\xleftarrow{k_B}$	Choose random $k_B \in \mathcal{K}_B$.
2.	Choose random $k_A \in \mathcal{K}_A$.		
	Compute $c = \mathbf{enclose}(k_A, k_B, b)$.	$\xrightarrow{c}$	c is commitment.
	To $\mathbf{open}(c)$:		
3.	Send k_A.	$\xrightarrow{k_A}$	Compute $b = \mathbf{reveal}(k_A, k_B, c)$. If $b = \phi$, then fail. If $b \neq \phi$, then b is revealed bit.

Figure 116.1: A generic bit commitment protocol.

1. $\forall k_A \in \mathcal{K}_A, \forall k_B \in \mathcal{K}_B, \forall b \in \{0,1\}$, $\mathbf{reveal}(k_A, k_B, \mathbf{enclose}(k_A, k_B, b)) = b$;
2. $\forall k_B \in \mathcal{K}_B, \forall c \in \mathcal{B}, \exists b \in \{0,1\}, \forall k_A \in \mathcal{K}_A$, $\mathbf{reveal}(k_A, k_B, c) \in \{b, \phi\}$.
3. No feasible probabilistic algorithm that attempts to distinguish blobs containing 0 from those containing 1, given k_B and c, is correct with probability significantly greater than 1/2.

The intention is that k_A is chosen by Alice and k_B by Bob. Intuitively, these conditions say:

1. Any bit b can be committed using any key pair k_A, k_B, and the same key pair will open the blob to reveal b.
2. For each k_B, all k_A that successfully open c reveal the same bit.
3. Without knowing k_A, the blob does not reveal any significant amount of information about the bit it contains, even when k_B is known.

A bit commitment scheme looks a lot like a symmetric cryptosystem, with $\mathbf{enclose}(k_A, k_B, b)$ playing the role of the encryption function and $\mathbf{reveal}(k_A, k_B, c)$ the role of the decryption function. However, they differ both in their properties and in the environments in which they are used. Conventional cryptosystems do not require condition 2, nor do they necessarily satisfy it. In a conventional cryptosystem, it is assumed that Alice and Bob trust each other and both share a secret key k. The cryptosystem is designed to protect Alice's secret message from a passive eavesdropper Eve. In a bit commitment scheme, Alice and Bob cooperate in the protocol but do not trust each other to choose the key. Rather, the key is split into two pieces, k_A and k_B, with each participant controlling one piece.

A bit commitment scheme can be turned into a bit commitment protocol by plugging it into the generic protocol given in Figure 116.1. The bit commitment protocol of section 113 ((lecture notes 23), and the protocols of sections 114 and 115 above can all be regarded as instances of the generic protocol. For example, we get the protocol of Figure 113.1 (lecture notes 23) by taking

$$\mathbf{enclose}(k_A, k_B, b) = E_{k_A}(k_B \cdot b),$$

and

$$\mathbf{reveal}(k_A, k_B, c) = \begin{cases} b & \text{if } k_B \cdot b = D_{k_A}(c) \\ \phi & \text{otherwise.} \end{cases}$$

117 Coin-Flipping

Alice and Bob are in the process of getting divorced and are trying to decide who gets custody of their pet cat, Fluffy. They both want the cat, so they agree to decide by flipping a coin: heads Alice wins; tails Bob wins. Bob has already moved out and does not wish to be in the same room with Alice. The feeling is mutual, so Alice proposes that she flip the coin and telephone Bob with the result.

This proposal of course is not acceptable to Bob since he has no way of knowing whether Alice is telling the truth when she says that the coin landed heads. "Look Alice," he says, "to be fair, we both have to be involved in flipping the coin. We'll each flip a private coin and XOR our two coins together to determine who gets Fluffy. You should be happy with this arrangement since even if you don't trust me to flip fairly, your own fair coin is sufficient to ensure that the XOR is unbiased." This sounds reasonable to Alice, so she lets him go on to propose the protocol of Figure 117.1. In this protocol, 1 means "heads" and 0 means "tails".

	Alice		Bob
1.	Choose random bit $b_A \in \{0,1\}$	$\xrightarrow{b_A}$.	
2.		$\xleftarrow{b_B}$	Choose random bit $b_B \in \{0,1\}$.
3.	Coin outcome is $b = b_A \oplus b_B$.		Coin outcome is $b = b_A \oplus b_B$.

Figure 117.1: Distributed coin flip protocol requiring honest parties.

After Alice considers Figure 117.1 for awhile, she objects. "This isn't fair. You get to see my coin flip before I see yours, so now you have complete control over the value of b." She suggests that she would be happy if the first two steps were reversed, so that Bob flipped his coin first, but Bob balks at that suggestion.

They then both remember last week's lecture and decide to use blobs to prevent either party from controlling the outcome. They agree on the protocol of Figure 117.2. At the completion of step 2, both Alice and Bob have each other's commitment (something they failed to achieve in the past, which is why they're in the middle of a divorce now), but neither know the other's private bit. They each learn each other's bit at the completion of steps 3 and 4.

While this protocol appears to be completely symmetric, it really isn't quite, for one of the parties completes step 3 before the other one does. Say Alice receives s_B before sending s_A. At that point, she can compute b_B and hence know the coin outcome b. If it turns out that she lost, she might decide to stop the protocol and refuse to complete her part of step 3.

We haven't really addressed the question for any of these protocols about what happens if one party quits in the middle or one party detects the other party cheating. We have only been concerned until now with the possibility of undetected cheating. But in any real situation, one party might feel that he or she stands to gain by cheating, even if the cheating is detected. That in turn raises complicated questions as to what happens next. Does a third party Carol become involved? If so, can Bob prove to Carol that Alice cheated? What if Alice refuses to talk to Carol? It may be instructive to think about the recourse that Bob has in similar real-life situations and to consider the reasons why such situations rarely arise. For example, what happens if someone fails to follow the provisions of a contract or if someone ignores a summons to appear in court?

There is another subtle problem with the protocol of Figure 117.2. Suppose that Bob sends his message before Alice sends hers in each of steps 1, 2, and 3. Then Alice can choose $k_A = k_B$,

	Alice		Bob
1.	Choose random $k_A, s_A \in \mathcal{K}_A$.	$\xleftrightarrow{k_A,\, k_b}$	Choose random $k_B, s_B \in \mathcal{K}_B$.
2.	Choose random bit $b_A \in \{0,1\}$.		Choose random bit $b_B \in \{0,1\}$.
	$c_A = \mathbf{enclose}(s_A, k_B, b_A)$.	$\xleftrightarrow{c_A,\, c_B}$	$c_B = \mathbf{enclose}(s_B, k_A, b_B)$.
3.	Send s_A.	$\xleftrightarrow{s_A,\, s_B}$	Send s_B.
4.	$b_B = \mathbf{reveal}(s_B, k_A, c_B)$.		$b_A = \mathbf{reveal}(s_A, k_B, c_A)$.
	Coin outcome is $b = b_A \oplus b_B$.		Coin outcome is $b = b_A \oplus b_B$.

Figure 117.2: Distributed coin flip protocol using blobs.

$c_A = c_B$, and $s_A = s_B$ rather than following her proper protocol. In step 4, Bob will compute

$$b_A = \mathbf{reveal}(s_A, k_B, c_A) = \mathbf{reveal}(s_B, k_A, c_B) = b_B,$$

so he won't detect that anything is wrong, and the coin outcome is $b = b_A \oplus b_B = b_A \oplus b_A = 0$. Hence, Alice can force the outcome to be 0 simply by playing copycat.

This problem is not so easy to overcome. One possibility is for both Alice and Bob to check after step 1 that $k_A \neq k_B$. That way, if Alice, say, plays copycat on steps 2 and 3, there is a good chance that

$$b_A = \mathbf{reveal}(s_A, k_B, c_A) \neq \mathbf{reveal}(s_B, k_A, c_B) = b_B.$$

However, depending on the bit commitment scheme, a difference in only one bit in k_A and k_B might not be enough to ensure that different bits are revealed.

A better idea might be to both check that $k_A \neq k_B$ after step 1 and then to use $h(k_A)$ and $h(k_B)$ in place of k_A and k_B, respectively, in the remainder of the protocol, where h is a hash function. That way, even a single bit difference in k_A and k_B is likely to be magnified to a large difference in the strings $h(k_A)$ and $h(k_B)$. This should lead to the bits $\mathbf{reveal}(s_A, h(k_B), c_A)$ and $\mathbf{reveal}(s_B, h(k_A), c_B)$ being uncorrelated, even if $s_A = s_B$ and $c_A = c_B$.

118 Locked Box Paradigm

Protocols for coin flipping and for dealing a poker hand from a deck of cards can be based on the intuitive notion of locked boxes. This idea in turn can be implemented using commutative cryptosystems.

118.1 Coin-flipping using locked boxes

We discussed the coin-flipping problem in section 117 and presented a protocol based on bit commitment. Here we present a coin-flipping protocol based on the idea of locked boxes.

- Imagine two sturdy boxes with hinged lids that can be locked with a padlock. Alice writes "heads" on a slip of paper and "tails" on another and places one of these slips in each box. She puts a padlock on each box for which she holds the only key. She then gives both locked boxes to Bob, in some random order.

- Bob cannot open the boxes and does not know which box contains "heads" and which contains "tails". He chooses one of the boxes and locks it with his own padlock, for which he has the only key. Now the box has two locks on it, one belonging to Alice and one to Bob. He gives the doubly-locked box back to Alice.

- Alice removes her lock and returns the box to Bob.
- Bob removes his lock, opens the box, and learns the outcome of the coin toss. He gives the slip of paper from the unlocked box back to Alice.
- Alice verifies that it is her slip of paper, with her handwriting on it, that she prepared at the beginning. She sends her key to Bob.
- Bob removes Alice's lock from the other box and verifies that she carried out her protocol correctly. (In particular, he checks that the slip of paper in the other box contains the other coin value.)

118.2 Commutative cryptosystems

Alice and Bob can carry out this protocol electronically using any *commutative* cryptosystem, that is, one in which $E_A(E_B(m)) = E_B(E_A(m))$ for all messages m. RSA is commutative for keys with a common modulus n, so we can use RSA in an unconventional way. Rather than making the encryption exponent public and keeping the factorization of n private, we turn things around. Alice and Bob jointly chose primes p and q, and both compute $n = pq$. Alice then chooses an RSA key pair $A = ((e_A, n), (d_A, n))$, which she can do since she knows the factorization of n. Similarly, Bob chooses an RSA key pair $B = ((e_B, n), (d_B, n))$ using the same n. Alice and Bob both keep their key pairs private (until the end of the protocol, when they reveal them to each other to verify that there was no cheating).

We note that this scheme may have completely different security properties from usual RSA. In RSA, there are three different secrets involved with the key: the factorization of n, the encryption exponent e, and the decryption exponent d. We have seen previously that knowing n and any two of these pieces of information allows the third to be reconstructed. Thus, knowing the factorization of n and e lets one compute d (easy). We also showed in section 56.3 ((lecture notes 13) how to factor n given both e and d.

The way RSA is usually used, only e is public, and it is believed to be hard to find the other quantities. Here we propose making the factorization of n public but keeping e and d private. It may indeed be hard to find e and d, even knowing the factorization of n, but if it is, that fact is not going to follow from the difficulty of factoring n. Of course, for security, we need more than just that it is hard to find e and d. We also need it to be hard to find m given $c = m^e \bmod n$. This is reminiscent of the discrete log problem, but of course n is not prime in this case.

118.3 Coin-flipping using commutative cryptosystems

Assuming RSA used in this new way is secure, we can implement the locked box protocol as shown in Figure 118.1. Here we assume that Alice and Bob initially know large primes p and q. In step (2), Alice chooses a random number r such that $r < (n-1)/2$. This ensures that m_0 and m_1 are both in $\mathbf{Z}_n$. Note that i and r can be efficiently recovered from m_i since i is just the low-order bit of m_i and $r = (m_i - i)/2$.

To see that the protocol works when both Alice and Bob are honest, observe that in step 3, $c_{ab} = E_B(E_A(m_j))$ for some j. Then in step 4, $c_b = D_A(E_B(E_A(m_j))) = E_B(m_j)$ by the commutativity of E_A and E_B. Hence, in step 5, $m = m_j$ is one of Alice's strings from step 2.

A dishonest Bob can control the outcome of the coin toss if he can find two keys B and B' such that $E_B(c_a) = E_{B'}(c'_a)$, where $C = \{c_a, c'_a\}$ is the set received from Alice in step 2. In this case, $c_{ab} = E_B(E_A(m_j)) = E_{B'}(E_A(m_{1-j}))$ for some j. Then in step 4, $c_b = E_B(m_j) = E_{B'}(m_{1-j})$.

	Alice		Bob
1.	Choose RSA key pair A with modulus $n = pq$.		Choose RSA key pair B with modulus $n = pq$.
2.	Choose random $r \in \mathbf{Z}_{(n-1)/2}$. Let $m_i = 2r + i$, for $i \in \{0, 1\}$. Let $c_i = E_A(m_i)$ for $i \in \{0, 1\}$. Let $C = \{c_0, c_1\}$.	$\xrightarrow{C}$	Choose $c_a \in C$.
3.		$\xleftarrow{c_{ab}}$	Let $c_{ab} = E_B(c_a)$.
4.	Let $c_b = D_A(c_{ab})$.	$\xrightarrow{c_b}$	
5.			Let $m = D_B(c_b)$. Let $i = m \bmod 2$. Let $r = (m - i)/2$. If $i = 0$ outcome is "tails". If $i = 1$ outcome is "heads".
		$\xleftarrow{B}$	
6.	Let $m = D_B(c_b)$. Check $m \in \{m_0, m_1\}$. If $m = m_0$ outcome is "tails". If $m = m_1$ outcome is "heads".		
		$\xrightarrow{A}$	
7.			Let $c'_a = C - \{c_a\}$. Let $m' = D_A(c'_a)$. Let $i' = m' \bmod 2$. Let $r' = (m' - i')/2$. Check that $i' \neq i$ and $r' = r$.

Figure 118.1: Distributed coin flip protocol using locked boxes.

Hence, $m_j = D_B(c_b)$ and $m_{1-j} = D_{B'}(c_b)$, so Bob can obtain both of Alice's messages and then send B or B' in step 5 to force the outcome to be as he pleases.

118.4 Card dealing using locked boxes

The same locked box paradigm can be used for dealing a 5-card poker hand from a deck of cards. Alice takes a deck of cards, places each card in a separate box, and locks each box with her lock. She arranges the boxes in random order and ships them off to Bob. Bob picks five boxes, locks each with his lock, and send them back. Alice removes her locks from those five boxes and returns them to Bob. Bob unlocks them and obtains the five cards of his poker hand. Further details are left to the reader.

119 Oblivious Transfer

In the locked box coin-flipping protocol, Alice has two messages m_0 and m_1. Bob gets one of them. Alice doesn't know which (until Bob tells her). Bob can't cheat to get both messages. Alice can't

cheat to learn which message Bob got. The *oblivious transfer problem* abstracts these properties from particular applications such as coin flipping and card dealing,

119.1 Oblivious transfer of a secret

Alice has a secret s. In an oblivious transfer protocol, half of the time Bob learns s and half of the time he learns nothing. Afterwards, Alice doesn't know whether or not Bob learned s. Bob can do nothing to increase his chances of getting s, and Alice can do nothing to learn whether or not Bob got her secret. Rabin proposed an oblivious transfer protocol based on quadratic residuosity, shown in Figure 119.1, in the early 1980's.

	Alice		Bob
1.	Secret s. Choose primes p, q, and let $n = pq$. Generate RSA public key (e, n). Compute $c = E_{(e,n)}(s)$.	$\xrightarrow{(e,n,c)}$	
2.		$\xleftarrow{a}$	Choose random $x \in \mathbf{Z}_n^*$. Compute $a = x^2 \bmod n$.
3.	Check $a \in \mathrm{QR}_n$. Choose y at random from among the 4 square roots of $a \pmod n$.	$\xrightarrow{y}$	
4.			Check $y^2 \equiv a \pmod n$. If $y \not\equiv \pm x \pmod n$, use x, y to factor n and decrypt c to obtain s.

Figure 119.1: Rabin's oblivious transfer protocol.

Alice can can carry out step 3 since she knows the factorization of n and can find all of the square roots of a. However, she has no idea which x Bob used to generate a. Hence, with probability $1/2$, $y \equiv \pm x \pmod n$ and with probability $1/2$, $y \not\equiv \pm x \pmod n$. If $y \not\equiv \pm x \pmod n$, then the two factors of n are $\gcd(x - y, n)$ and $n/\gcd(x - y, n)$, so Bob factors n and decrypts c in step 4. However, if $y \equiv \pm x \pmod n$, he learns nothing, and Alice's secret is as secure as RSA itself.

There is one potential problem with this protocol. A cheating Bob in step 2 might send a number a which he generated by some other means than squaring a random x. In this case, he learns something new no matter which square root Alice sends him in step 3. Perhaps that information, together with what he already learned in the course of generating a, is enough for him to factor n. We don't know of any method by which Bob could find a quadratic residue without also knowing one of its square roots. We certainly don't know of any method that would produce a quadratic residue a and some other information Ξ that, combined with y, would allow Bob to factor n. But we also cannot prove that no such method exists.

We can fix this protocol by inserting between steps 2 and 3 a zero knowledge proof that Bob knows a square root of a. This is essentially what the simplified Feige-Fiat-Shamir protocol does, but we have to reverse the roles of Alice and Bob. Now Bob is the one with the secret square root x. He wants to prove to Alice that he knows x, but he does not want Alice to get any information

about x, since if she learns x, she could choose $y = x$ and reduce his chances of learning s while still appear to be playing honestly. Again, details are left to the reader.

119.2 One-of-two oblivious transfer

In the *one-of-two oblivious transfer*, Alice has two secrets, s_0 and s_1. Bob always gets exactly one of the secrets. He gets each with probability 1/2, and Alice does not know which he got.

The locked box protocol is one way to implement one-of-two oblivious transfer. Another is based on a public key cryptosystem (such as RSA) and a symmetric cryptosystem (such as AES). This protocol, given in Figure 119.2, does not rely on the cryptosystems being commutative.

	Alice		Bob
1.	Choose two PKS key pairs (e_0, d_0) and (e_1, d_1).	$\xrightarrow{e_0, e_1}$	
2.			Generate random key k for a symmetric cryptosystem $(\hat{E}, \hat{D})$. Choose random $b \in \{0, 1\}$.
		$\xleftarrow{c}$	Compute $c = E_{e_b}(k)$.
3.	Compute $k_i = D_{d_i}(c)$ for $i = 0, 1$. Choose $b' \in \{0, 1\}$. Compute $c_i = \hat{E}_{k_i}(s_{i \oplus b'})$ for $i = 0, 1$.	$\xrightarrow{c_0, c_1}$	
4.			Output $s = s_{b \oplus b'} = \hat{D}_k(c_b)$.

Figure 119.2: One-of-two oblivious transfer using two PKS key pairs.

In step 2, Bob encrypts a randomly chosen key k for the symmetric cryptosystem using one of the PKS encryption keys that Alice sent him in step 1. In step 2, Bob selects one of the two encryption keys from Alice, uses it to encrypt k, and sends the encryption to Alice. In step 3, Alice decrypts c using both decryption keys d_0 and d_1 to get k_0 and k_1. One of the k_i is Bob's key k (k_b to be specific) and the other is garbage, but because k is random and she doesn't know b, she can't tell which is k. She then encrypts one secret with k_0 and the other with k_1, using the random bit b' to ensure that each secret is equally likely to be encrypted by the key that Bob knows. In step 4, Bob decrypts the ciphertext c_b using key his key $k = k_b$ to recover the secret $s = s_{b \oplus b'}$. He can't decrypt the other ciphertext $c_{1 \oplus b}$ since he doesn't know the key $k_{1 \oplus b}$ used to produce it, nor does he know the decryption key $d_{1 \oplus b}$ that would allow him to find it from c.