6.875: Solutions for Homework 7

Yevgeniy Dodis

Problem 1: El-Gamal Signature

Here is a very popular signature scheme due to El-Gamal. The public key consists of a prime p, a generator g for $\mathbb{Z}_p^*$ and some element $y \in \mathbb{Z}_p^*$. The secret key consists of an integer $x \in \{1 \dots (p-1)\}$ such that $y = g^x \bmod p$.

To sign a message m, the user picks a random $k \in \{1 \dots (p-1)\}$ such that $gcd(k, p-1) = 1$, computes $r = g^k \bmod p$, and then solves for s the linear equation $m = xr + ks \bmod (p-1)$ (which is possible to do since k is relatively prime to $(p-1)$). The signature is a pair $\langle r, s \rangle$. To verify a signature $\langle r, s \rangle$ of m, we check that $g^m = y^r r^s \bmod p$.

(a) Show that El-Gamal signature is existentially forgeable under key only attack, i.e. it is possible to generate a valid message/signature pair by knowing only y, p and g! If you don't succeed, show existential forgery if a single valid message/signature pair $(m, \langle r, s \rangle)$ is known.

Solution: There are ton of ways to proceed. In general, it's a nice problem for hackers giving no conceptual insight. Here is the most trivial solution. Set $m = p-1$, $r = p-1$, $s = 0$. Then $g^m = g^{p-1} = 1 = 1 \cdot (p-1)^0 = y^{p-1}(p-1)^s = y^r r^s$. Similarly for $m = (p-1)/2$, $r = p-1$, $s = 1$. A more interesting way is to pick $r = g^a y^b \bmod p$, $s = -rb^{-1} \bmod (p-1)$, $m = -rab^{-1} \bmod (p-1)$, where $a \in \mathbb{Z}_{p-1}$ and $b \in \mathbb{Z}_{p-1}^*$. Then

$$y^r r^s = y^r (g^a y^b)^s = y^{(r+bs) \bmod (p-1)} g^{as \bmod (p-1)} = y^0 g^m = g^m \mod p$$

An interesting special case a lot of you observed is when $b = 1$, i.e. $r = g^a y \bmod p$, $s = -r \bmod (p-1)$, $m = -ra \bmod (p-1)$ (and $a \in \mathbb{Z}_{p-1}$). Other examples are possible, just be careful when dealing modulo p (in the base) and modulo $(p-1)$ (in the exponent).

(b) Show that El-Gamal signature scheme can be completely broken when the same (still random) integer k is used to compute the signatures of two distinct messages m_0 and m_1.

Solution: Given 2 signatures $\langle r_0, s_0 \rangle$, $\langle r_1, s_1 \rangle$ of m_0 and m_1 corresponding to the same k (i.e. $r_0 = r_1 = g^k = r \bmod p$), we get the following system with 2 equations and 2 unknowns (k and x) modulo $(p-1)$:

$$\begin{aligned} m_0 &= xr + ks_0 \\ m_1 &= xr + ks_1 \end{aligned}$$

Easy manipulations imply that

$$m_0 s_1 - m_1 s_0 = xr(s_1 - s_0) \bmod (p-1)$$

If r and $(s_1 - s_0)$ are relatively prime to $(p-1)$, we completely break the system since $x = (m_0 s_1 - m_1 s - 0) r^{-1} (s_1 - s_0)^{-1} \bmod (p-1)$. Can we ensure that r and $(s_1 - s_0)$ are in $\mathbb{Z}_{p-1}^*$? You did not have to do it, but the answer is more or less yes. First of all, from the 2 main equations above $k(s_1 - s_0) = m_1 - m_0$, and since $gcd(k, p-1) = 1$, $gcd(s_0 - s_1, p-1) = 1$ iff $gcd(m_1 - m_0, p-1) = 1$. Since we are free to choose m_0 and m_1, we can make $gcd(m_1 - m_0, p-1) = 1$.

What about $r = g^k \bmod p$, where k is random such that $gcd(k, p-1) = 1$? Since g is a generator and $gcd(k, p-1) = 1$, we get that r is a *random generator* of $\mathbb{Z}_p^*$. Hence, the question is: what is the probability that a random generator r of $\mathbb{Z}_p^*$ is relatively prime to $p-1$? This is a technically non-trivial question, requiring some analytic number theory. Intuitively, we expect that the fact that r has to be a generator "has nothing to do" with being relatively prime to $p-1$. If r was random over $\mathbb{Z}_p^*$, the answer would be exactly $\varphi(p-1)/(p-1)$ (which by another classical result is at least $1/\operatorname{loglog}(p-1)$). In fact, using some heavy machinery (like the Moebius invertion formula), one can show that this is more or less the right answer, i.e. our probability is $\frac{\varphi(p-1)}{(p-1)} + o(\frac{1}{\sqrt{p}}) \geq \frac{1}{\operatorname{loglog}(p-1)}$, which is quite non-negligible making the system insecure indeed.

(c) What about picking the initial k at random, but then using $k+2$, $k+4$, etc. to sign subsequent messages (assume that $gcd(k+2j, p-1) = 1$ for at least several consecutive $j \geq 0$)?

Solution: Same idea as before except I won't go in detail ensuring that appropriate numbers can be made relatively prime to $(p-1)$. Again for any two indices i and j (say, $i = 0,\ j = 1$), we get a linear system modulo $(p-1)$ (here $r_i = g^{k+2i} \bmod p$):

$$\begin{aligned} m_i &= xr_i + (k+2i)s_i \\ m_j &= xr_j + (k+2j)s_j \end{aligned}$$

From here,

$$m_i s_j - m_j s_i + 2(j-i)s_i s_j = x(r_i s_j - r_j s_i) \bmod (p-1)$$

which can be solved for x assuming the appropriate inverse exists modulo $(p-1)$.

Problem 2: Batch Verifier for RSA

Assume we are given a signature scheme such that the message/signature pairs always comes in batches $(m_1, s_1), \ldots, (m_k, s_k)$ of size k. One way is to apply k independent verification procedures for each (m_i, s_i). However, we might hope to make a *batch verifier* that works significantly faster. More formally, a batch verifier BV is a PPT algorithm that gets a public key, k message/signature pairs $\{(m_i, s_i)\}_{i=1}^k$ and

- If all signatures are valid, BV accepts with probability 1.
- If at least one signature is invalid, BV accepts with probability at most 2^{-l}, where l is the security parameter and the probability is *only over the coin tosses of BV*.

(a) The following is a proposed batch verifier for the RSA signature scheme. Given a public key (n, e) and a vector of pairs $\{(m_i, s_i)\}_{i=1}^k$, we compute $m = \prod_i m_i \bmod n,\ s = \prod_i s_i \bmod n$, and check if $s^e = m \bmod n$.

Is it a good batch verifier for RSA? If yes, determine its security parameter l. Otherwise, explain why not.

Solution: This is a terrible batch verifier. Take any $s_1, \ldots, s_k$, let $m_1 = m_2 = \ldots = m_{k-1} = 1$, $m_k = (\prod_i s_i)^e$. This passes the test while none of the s_i's is a correct signature for the corresponding message m_i.

(b) Here is another suggestion for the RSA signature when e is large. It was suggested by Bellare, Garay and Rabin.

1. Repeat l times:
2. Pick a random subset $I \subseteq \{1 \ldots k\}$.
3. Compute $m_I = \prod_{i \in I} m_i \bmod n$, $s_I = \prod_{i \in I} s_i \bmod n$.
4. If $s_I^e \neq m_I \bmod n$, output 0 and halt.
5. Output 1 iff all tests succeed.

Prove that the above is indeed a good batch verifier for RSA with security parameter l. Let multiplication take 1 unit of time. Assume exponentiation to the power x takes $3|x|/2$ units (multiplications). For which settings of l and $|e|$ is the above batch verifier better than the naive one that verifies each signature separately?

Solution: Clearly, if all signatures are valid, the test accepts with probability 1 since for any I, $s_I^e = \prod_{i \in I} s_i^e = \prod_{i \in I} m_i = m_I$. Assume at least one of the message/signature pairs is wrong, say (m_i, s_i). We claim that each of the l iterations has probability of at least 1/2 of outputting 0, justifying the fact that this is a good batch verifier with security parameter l. For that, take any set I such that $i \notin I$ and let $I' = I \cup \{i\}$. It cannot be the case that both I and I' pass the test, i.e. $m_I = s_I^e$ and $m_{I'} = s_{I'}^e$ since this would imply that $m_i = s_i^e$. Since we pick a random subset of $\{1 \ldots k\}$, the pairs $\{(I, I') \mid i \notin I\}$ perfectly split the set of subsets of $\{1 \ldots k\}$ and at least one subset in each pair makes the test fail, each iteration succeeds with probability at most 1/2 indeed.

The trivial scheme takes k exponentiations, i.e. $3k|e|/2$ units. Since we expect $|I| \approx k/2$, computing m_I and s_I together takes k units on average. Then we make 1 exponentiation, a total of $(k + 3|e|/2)$ units per iteration. Counting l iterations, this test beats the trivial one when $l(k + \frac{3|e|}{2}) < \frac{3k|e|}{2}$. Asymptotically, we need $l \ll \min(k, |e|)$, i.e. the RSA exponent and the size of the batch should be large.

Problem 3: Attacking Blum-Goldwasser Cryptosystem

Recall the Blum-Goldwasser public key encryption scheme that is GM-secure if factoring Blum integers is hard. The public key is n which is a Blum integer. To encrypt m of length k, we create a "one-time pad" $p = p_1 \ldots p_k$ by picking a random square x and letting $p_i = lsb(x^{2^{i-1}} \bmod n)$. The encryption is $c = \langle m \oplus p, x^{2^k} \bmod n \rangle$. To decrypt $\langle R, y \rangle$, we let y' be the principal square root of y^2 (i.e. we interpret y as a quadratic residue), reconstruct x by taking k principal square roots of y', reconstruct p from x, and finally let $m = p \oplus R$.

Assume we augment the capabilities of the forger F. After a random public/secret key pair is chosen and the public key is given to F, rather than asking F to right away distinguish encryptions of m_0 and m_1 (in three-pass, say), we first give him some partial access to the decryption algorithm as described below. Only after the adversary declares that he is done (adaptively) using the decryption algorithm (any polynomial number of times), we proceed as before challenging him to distinguish encryptions of m_0 and m_1 (now F no longer has this partial access).

For each of the following attacks, determine (with proof) if Blum-Goldwasser encryption scheme is still GM-secure.

(a) *Chosen ciphertext attack*: for any string c of F's choice, we tell F the correct decryption m of c (or that c is not a valid encryption of any message).

(b) *Replay attack*: for any two strings c_1 and c_2 of F's choice, we tell F if they are encryptions of the same message (without telling the message itself).

(c) *Correspondence attack*: for any strings m and c of F's choice, we tell F if c decrypts to m.

(d) *Validity attack*: for any string c of F's choice, we tell F if c is a valid encryption of some message (without telling the message itself).

(e) (Optional) Blum-Goldwasser is a sub-case of a general procedure for creating a GM-secure encryption based on any family of trapdoor permutations (where $f(x) = x^2 \bmod n$, $x \in QR(n)$, and $B(x) = lsb(x)$ is a hardcore bit for f). Take the weakest of the above attacks for which Blum-Goldwasser is insecure, if any. What general property of a trapdoor function f would make the encryption scheme insecure against this attack, like it did for the Blum-Goldwasser case.

Solution: The problem it is much easier than it sounds. Let $PS(z)$ denote the principal square root of $z \in QR(n)$ (i.e. the unique square root that is itself a square). Let us do the easy part (d) first.

(d) Looking at the decryption process of $\langle R, y\rangle$, we see that it *never fails*. Indeed, even if y is not quadratic residue, we compute $y' = PS(y^2)$, then we can extract k principal square roots of y' since $f(z) = z^2 \bmod n$ is a permutation over $QR(n)$ (for n – Blum integer). Then we always can get p and $m = p \oplus R$. So the attack gives no information whatsoever (F gets a "yes" for every c). Since the original scheme is GM-secure, it remains so.

(a,b,c) We show that any of these attacks allows to completely break the scheme by factoring n. First of all, it suffices to show it for part (c) only (correspondence attack), since replay and chosen ciphertext attacks are only stronger. Indeed, if one has a chosen ciphertext attack, he can simulate the replay attack by actually asking the decryptions of both c_1 and c_2, and then comparing them. If one has a replay attack, he can simulate the correspondence attack by encrypting m into, say, c' (remember, we have the public key) and then asking if c and c' decrypt to the same message (if they do, this message must be m itself; otherwise, c does not decrypt to m since c' does).

Second of all, we know that factoring is equivalent to taking square roots of arbitrary quadratic residues modulo n. Finally, we know that extracting square roots when n is a Blum integer is equivalent to finding the least significant bit of the principal square root of any quadratic residue (since $lsb(x)$ is a hardcore bit for $x^2 \bmod n$ over $QR(n)$). Putting it all together, it is sufficient to show how the correspondence attack allows us to obtain $lsb(PS(z))$ for any $z \in QR(n)$ (then we can extract square roots and factor n, completely breaking the scheme).

But this is simple now. Assume we are given $z \in QR(n)$. Let $m = 0^k$. Assume we are encrypting m starting from $x = PS(z)$ (we can't completely do it since we don't have x but we pretend we do). Then $p_1 = lsb(x) = lsb(PS(z))$, i.e. p_1 is exactly what we want to find somehow using the correspondence attack. But notice that for $i \geq 2$, $p_i = lsb(x^{2^{i-1}}) = lsb(z^{2^{i-2}})$. Hence we can compute by ourselves all the bits of p except the first bit p_1 (which we are trying to determine). Well, let us just set $p_1 = 0$ for now and use the correspondence attack to see if we are right! Set $y = x^{2^k} = z^{2^{k-1}} \bmod n$, which we can again compute by ourselves. Let F ask the partial decryptor whether $\langle 0p_2 \dots p_k, y\rangle$ is a valid encryption of $m = 0^k$. We know that it is iff $lsb(x)$ is actually 0. But then we are done since the answer tells us the correct $p_1 = lsb(PS(z))$.

(e) Tracing back how we used the correspondence attack, we see that for any x, it allows us to compute (always correctly) the hardcore bit $B(x)$ from $f(x)$. What we want is to completely break the scheme, i.e. find the trapdoor. So the first sufficient general property for f is: being able to compute $B(x)$

from $f(x)$ allows us to to obtain the trapdoor. Not much more can be said in general, but if we use using the Goldreich-Levin bit $B(x, r)$, we know that the ability to compute $B(x, r)$ implies the ability to invert f with non-negligible (actually very close to 1 if we can always invert B) probability (taken over the inputs to f). So in this case the property would be that inverting f with non-negligible probability allows one to obtain the trapdoor. Another further specialization would be look at "homomorphic" f where the ability to invert at a random point implies the ability to invert at *any* point (like RSA or Rabin). Then the property would be further refined to saying that the ability to invert f at arbitrary point implies the ability to obtain the trapdoor. For our case this says that extracting (principal) square roots is equivalent to factoring, which is true. But even for the RSA this property it is not clear and is a major open problem: is the ability to take e-th roots modulo n equivalent to factoring? Most people believe that "no", so such a simple attack might not break the GM-secure scheme based on the RSA family.

Problem 4: Zero-Knowledge Proof Systems

Let $L = \{(p, g) \mid p \text{ - prime}, g \text{ - generator for } \mathbb{Z}_p^*\}$. Design a *zero-knowledge* interactive proof system for L. That is, when $(p, g) \in L$, a PPT verifier V should be convinced by an honest (maybe computationally unbounded) prover P that this is indeed so (say, with probability 1). However, P should do it in such a way that V (or any PPT verifier $\tilde{V}$ for this matter!) does not learn anything else other than "p is a prime and g is a generator". On the other hand, if $(p, g) \notin L$, no (even computationally unbounded and potentially cheating) prover $\tilde{P}$ should be able to convince V that $(p, g) \in L$ with probability more than 1/2. The formal general definition follows below.

Definition 1 (Zero-Knowledge Interactive Proof System (ZKIP)) *A language $L \subseteq \{0,1\}^*$ is said to belong to class* **ZKIP** *if there exists a PPT verifier V and a (possibly computationally unbounded) prover P such that (the probabilities below are taken over the random coins of the participating parties, but not over the input x which is fixed)*

- *(Completeness) If $x \in L$, then* $\Pr[\ [P \leftrightarrow V](x) = 1\ ] = 1$ *(i.e. V always accepts).*
- *(Soundness) If $x \notin L$, then for any $\tilde{P}$,* $\Pr[\ [\tilde{P} \leftrightarrow V](x) = 1\ ] \leq \frac{1}{2}$.
- *(Zero-Knowledge) There exists a PPT* simulator *S such that for any PPT verifier $\tilde{V}$, if we denote by $S^{\tilde{V}}(x)$ running S on x with a black-box access*[1] *to $\tilde{V}$, we have that for any $x \in L$,*

$$\mathrm{VIEW}_{\tilde{V}}([P \leftrightarrow \tilde{V}](x)) \approx S^{\tilde{V}}(x) \tag{1}$$

Here $\mathrm{VIEW}_B([A \leftrightarrow B](x))$ denotes the random variable consisting of B's view of the interaction on input x, i.e. B's random coins together with all the messages sent by both players (technically speaking, it suffices to specify only messages that B received from A, but it does not matter if we include all the messages).

For the purposes of this homework, the $\approx$ "equivalence" sign above means that the two distributions in question are identical[2].

Solution: Let us first describe the correct protocol for honest P and V on input p and g. Let $ind(x)$ denote the index (i.e. the discrete log) of x base g, if it exists.

[1] This means S can run $\tilde{V}$ with any random coins, any input and starting from any valid configuration. In particular, it can reset $\tilde{V}$ to some previous state and start running it from there again. $\tilde{V}$ will stop at the next point where he is supposed to receive a message. S gets all the messages sent by $\tilde{V}$ from the point of invocation to the point where he stopped waiting for the next message, as well as the final state of $\tilde{V}$, so that S can continue running $\tilde{V}$ later.

[2] Such **ZKIP** is called *perfect*. It is also possible to talk about *statistical* and *computational* zero-knowledge.

- V checks if p is a prime. If not, he rejects. Else, V picks a random x in $\mathbb{Z}_p^*$ and sends x to P.
- P stops if $x \notin \mathbb{Z}_p^*$. Else P picks a random y in Z_p^*, and sends y to V.
- V flips a coin c and sends c to P.
- P stops if c is not a bit. If $c = 0$, P sets $z = ind(y)$, else $z = ind(xy)$. P sends z to V.
- V verifies the result by checking that $g^z = z$ when $c = 0$, or that $g^z = xy$ when $c = 1$.

Let's start to argue that this is a correct protocol. Completeness is easy. If p is a prime and g is a generator, P can find the index of both y and xy (since both exist and P is infinitely powerful). Let's argue the soundness of 3/4. For that assume $(p, g) \notin L$ and take any (maybe cheating) $\tilde{P}$. If p is not a prime, V will immediately reject so we assume that p is a prime, so that g is *not* a generator as $(p, g) \notin L$. Since g is not a generator and $\mathbb{Z}_p^*$ is a group of size $(p-1)$, the order of g must be a non-trivial divisor of $(p-1)$, i.e. at most $(p-1)/2$. Thus, at most half of the elements of $\mathbb{Z}_p^*$ have a discrete log base g. Since x was chosen at random by an honest V, x has discrete log base g with probability at most 1/2. Also, for any y, both y and xy have a discrete log base g only if x has a discrete log base g (since $ind(x) = ind(xy) - ind(y)$, if both indices exist). Hence, with probability $\geq 1/2$ at least one of y or xy does not have a discrete log base g. Assume this is so for now. Then with probability at least 1/2 (already conditioned on the fact that at least one of x, xy does not have an index base g) an honest V will choose c such that he will challenge $\tilde{P}$ with the discrete log of whichever one of y or xy that does not have a discrete log. Hence, with probability at least $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$, $\tilde{P}$ would not be able to answer, so we get soundness of 3/4 as claimed. But repeating the protocol *serially* (parallel composition does not work) 3 times, we reduce soundness to $(\frac{3}{4})^3 < \frac{1}{2}$.

It remains to argue the zero-knowledge. Intuitively, our protocol "forces" V (or any $\tilde{V}$) to learn a discrete log of a random number: either y or xy, where y was chosen by the (now honest) prover P. But this gives $\tilde{V}$ no information (other than g is a generator) since he can generate a discrete log of a random number on his own. Let's argue it formally by describing a PPT simulator $S^{\tilde{V}}$. For this part, we can assume that $(p.g) \in L$, i.e. p is a prime and g is a generator. Here is $S^{\tilde{V}}(p, g)$.

1. Flip random coins r for $\tilde{V}$.
2. Run $\tilde{V}(p, g)$ giving him r as his coins and get x from $\tilde{V}$. If $\tilde{V}$ stops instead, output $\langle r, ()\rangle$ and halt.
3. If $x \notin \mathbb{Z}_p^*$, output $\langle r, (x)\rangle$ and halt.
4. Pick a random $z \in \mathbb{Z}_p^*$ and flip a coin c'. If $c' = 0$ then set $y = g^z$, else set $y = x^{-1}g^z$.
5. Run $\tilde{V}$ from the previous state (i.e. same coins, same x) by giving y as a message from P. Wait for response c. If $\tilde{V}$ stops instead, output $\langle r, (x, y)\rangle$ and halt.
6. If c is not a bit, output $\langle r, (x, y, c)\rangle$ and halt.
7. If $c \neq c'$, go back to step 2. Otherwise, output $\langle r, (x, y, c, z)\rangle$ and halt.

Aside from silly house-keeping (that I was so careful about for completeness), the crucial steps are steps 4 and 7. In step 4, the simulator prepared the responses y and z by going "backwards", but z is valid only for a randomly chosen bit c'. Indeed, if $c' = 0$, we have $y = g^z$, else we have $xy = g^z$. So we hope that $\tilde{V}$, no matter how malicious and ill-willed he is, would produce $c = c'$. If we are unlucky, we simply start from scratch by going back to step 2. Otherwise, we indeed produce the view of $\tilde{V}$ that is *completely identical* with his view when talking to real P. Indeed, his coins r are truly random, x he generated himself, y is the random element of $\mathbb{Z}_p$ irrespective of c' and z (indeed, both g^z and $x^{-1}g^z$ are random for a random z, so it is exactly what P would have sent), c he generated himself again and, finally, z is the correct answer (exactly as P would give it). Hence, we only have to argue that we will be lucky very soon. But as we pointed out, y is random irrespective of c' (i.e. y and c' are *independent*), so the knowledge of y gives $\tilde{V}$ no information about c'. Hence, no matter how he produces c', $\Pr[c' = c] = \frac{1}{2}$, so we expect to be done in 2 iterations. By Chernoff bound, if you wish, the probability that we do not succeed in having $c' = c$ in $O(\log n)$ iterations is exponentially small. Hence, S is indeed PPT and produces output identical to the view of $\tilde{V}$ when talking to real P. This completes the proof.

Typical Wrong Proofs: Here are two examples of the wrong proofs, showing that one has to be careful.

- V checks if p is a prime. If not, he rejects. Else, V picks a random y in $\mathbb{Z}_p^*$ and sends y to P.
- P stops if $y \notin \mathbb{Z}_p^*$. P finds $x = ind(y)$ and sends x to V.
- V verifies that $g^x = y \bmod p$.

While completeness and soundness are easy to establish (i.e. this is a valid interactive proof system for L), zero-knowledge does *not* hold. In fact, it does hold against the *honest* verifier V who really chooses a *random* y. Indeed, a simulator can simply choose a random x, set $y = g^x$ and output the corresponding view of V. However, a cheating verifier $\tilde{V}$ can learn something that he "cannot come up on his own" by picking some fixed (not random) y (e.g. he sets $y = 2$ and learns the discrete log of 2, something that he probably cannot compute himself even when knowing that g is indeed a generator). Indeed, we immediately run into a problem when trying to construct a simulator S. After S asks $\tilde{V}$ about y, there is no way for S to compute the index x of y, since S cannot take discrete logs and since y could have be chosen maliciously by $\tilde{V}$ (so that S can't "prepare" the answer beforehand). This shows why it is so important that the zero-knowledge holds against *any* $\tilde{V}$.

A more subtle example of a wrong protocol is the following (due to Michael Kaminskiy, Eric Kuo and Qixiang Sun)

- V checks if p is a prime. If not, he rejects.
- P picks a random $a \in \mathbb{Z}_{p-1}^*$, and sends $x = g^a$ to V.
- V picks a random b in $\mathbb{Z}_{p-1}$, and sends $y = x^b$ to P.
- P computes $z = ind(y) \cdot a^{-1} \bmod (p-1)$, and sends z to V.
- V verifies that $b = z$.

It is less obvious, but this protocol is still complete and sound. Indeed, if g is a generator, then $x = g^a$ is *also* a generator (in fact, a random generator) of $\mathbb{Z}_p^*$, since $gcd(a, p-1) = 1$. Since $y = x^b = g^{ab}$, we have that $ind(y) = ab$, so indeed $b = ind(y)a^{-1} \bmod (p-1)$. If g is not a generator, then there are exactly $\frac{p-1}{order(g)} \geq 2$ solutions $k \in \mathbb{Z}_{p-1}$ to the equation $g^{ab} = g^k \bmod p$. Hence, there are at least 2 values b' such that $y = g^{ab'}$. Since P only knows y, the probability it correctly guesses $b' = b$ is at most $\frac{1}{2}$.

However, the protocol is still not zero-knowledge, at least it is not known how to show that it is (again, it is easy against honest V by choosing random a and b and outputting the appropriate view, but cheating $\tilde{V}$ is a problem). While when g is a generator, $\tilde{V}$ seems to only learn a discrete log of a "random" element y, we cannot *prove* it. The problem is that $\tilde{V}$ chooses y *after* he sees g^a. It is conceivable that $\tilde{V}$ can learn some partial information about the structure of a from g^a (for example, it *does* learn the $lsb(a)$). Then he picks some malicious y (maybe without choosing any b and maybe totally unrelated to x). And maybe

$z = ind(y)a^{-1} \bmod (p-1)$, together with the partial information about a that $\tilde{V}$ had, enable $\tilde{V}$ to obtain some partial information about the index of y, something $\tilde{V}$ might not be able to do on his own.

Of course, this was not a formal proof, but it gave an intuition why the simulation attempts fail. If we try to construct a simulator S, after S learns y there seem to be no way it can (prepare beforehand to) compute $z = ind(y) \cdot a^{-1}$, as the true prover P would have done, since he does not know how y was really chosen by $\tilde{V}$.