More on safe primes

Recall that p is a **safe prime** if both p and $(p-1)/2$ are prime. The next example illustrates why it is common to use safe primes for ElGamal.

In general, it is difficult to ensure that g is a primitive root, or almost a primitive root, modulo p.

Example 179. Suppose that p is a safe prime. Show that all residues $g \not\equiv 0, \pm 1 \pmod{p}$ have order $(p-1)/2$ or $p-1$.

In the latter case, g is a primitive root. In fact, if $p > 5$, then half of the residues $g \not\equiv 0, \pm 1$ are primitive roots.

Solution. Suppose $g \not\equiv 0, \pm 1 \pmod{p}$. Because p is a prime and $g \not\equiv 0$, g is invertible. Its multiplicative order N divides $\phi(p) = p-1$. But the prime factorization of $p-1$ is 2 times $(p-1)/2$. Hence, the only possible orders are $1, 2, (p-1)/2$ and $p-1$. The residues ± 1 are the only with order 1 and 2 (why?!). Thus, g must have order $(p-1)/2$ or $p-1$.

Finally, if $p > 5$ (so that $(p-1)/2$ cannot be 2 and so must be odd since we assume it is a prime), note that the number of primitive roots is $\phi(p-1) = \phi(2)\phi((p-1)/2) = (p-3)/2$, which is exactly half of the remaining residues g (without $0, \pm 1$).

Comment. Recall that, if x has order $p-1$, then x^2 has order $\frac{p-1}{\gcd(p-1,2)} = \frac{p-1}{2}$. It follows that quadratic residues are those that have order $(p-1)/2$ (provided that $x \not\equiv 0, \pm 1$).

Example 180. Is there any advantage for RSA if p is a safe prime? Potential issues?

Solution. If p is a safe prime, then $\gcd(p-1, q-1) = 2$. Why?!

Hence, the key space is as large as possible.

On the other hand, we need to think about whether we are weakening the security in case we might severely limit the number of possible p's to choose from. [The prime number theorem tells us that this is not something we need to be too worried about. Can you spell out the details?]

Another issue is that generating random safe primes is considerably more work. On the other hand, Bob usually does not generate a public key frequently, so that this might not be much of an issue.

Armin Straub
straub@southalabama.edu

Further comments on RSA and ElGamal

Theorem 181. Determining the secret private key d in RSA is as difficult as factoring N.

Proof. Let us show how to factor $N = pq$ if we know e and d such that $d \equiv e^{-1} \pmod{(p-1)(q-1)}$.

- Write $ed - 1 = 2^t m$, where t is chosen as large as possible such that 2^t divides $ed-1$.
  Since $ed - 1 \equiv 0 \pmod{(p-1)(q-1)}$ and 2^2 divides $(p-1)(q-1)$, we have $t \geqslant 2$.
- Pick a random invertible residue x. Observe that $x^{ed-1} \equiv 1 \pmod{N}$. In other words, $(x^m)^{2^t} \equiv 1$.
  Hence, the multiplicative order of x^m must divide 2^t.
- Suppose that x^m has different order modulo p than modulo q.

  Note. One can show that this works for at least half of the (invertible) residues x. As such, if we are unlucky, we just select another x.

  Since both orders must divide 2^t, we may suppose x^m has order 2^s modulo p, and larger order modulo q.
  Then, $x^{2^s m} \equiv 1 \pmod{p}$ but $x^{2^s m} \not\equiv 1 \pmod{q}$.
  Consequently, $\gcd(x^{2^s m} - 1, N) = p$ so that we have found the factor p of N.

  Note. Of course, we don't know s (because we don't know p and q), but we can just go through all $s = 0, 1, 2, \ldots, t-1$. One of these has to reveal the factor p. □

However. It is not known whether knowing d is actually necessary for Eve to decrypt a given ciphertext c. This remains an important open problem.

Advanced comment. Recall that we don't necessarily need to have $d \equiv e^{-1} \pmod{(p-1)(q-1)}$ but that it suffices to have the same with $(p-1)(q-1)$ replaced by $\operatorname{lcm}(p-1, q-1)$. This means that, in the above argument, we only get $t \geqslant 1$ but the rest of the argument still applies.

Example 182. (homework) Bob's public RSA key is $N = 323$, $e = 101$. Knowing $d = 77$, factor N using the approach of the previous theorem.

Solution. Here, $de - 1 = 7776 = 2^5 \cdot 243$ so that $t = 5$ and $m = 243$.

- Let's pick $a = 2$. $a^m = 2^{243} \equiv 246 \pmod{323}$ must have order dividing 2^5.
  $\gcd(246^2 - 1, 323) = 19$ (so we don't even need to check $\gcd(246^{2^s} - 1, 323)$ for $s = 2, 3, 4$)
  Hence, we have factored $N = 17 \cdot 19$.

Comment. Among the $\phi(323) = 16 \cdot 18 = 288$ invertible residues a, only 36 would not lead to a factorization. The remaining 252 residues all reveal the factor 19.

Another project idea. Run some numerical experiments to get a feeling for the number of residues that result in a factorization.

Certain attacks on RSA and ElGamal

Example 183. What is your feeling? Can we make RSA even more secure by allowing N to factor into more than 2, say, 3 primes?

Solution. That doesn't seem like a good idea. Namely, observe that the security of RSA relies on adversaries being unable to factor N. Allowing more factors of N (while keeping the size of N fixed) makes that task easier, because more factors means that the factors are necessarily smaller.

Example 184. RSA has proven to be secure so far. However, it is easy to implement RSA in such a way that it is insecure. One important but occasionally messed up part of RSA is that **p and q must be unpredictable**, and the only way to achieve that is to choose p, q completely randomly in some huge interval $[M_1, M_2]$.

- For instance, if $N = pq$ has m digits and we know the first (or last) $m/4$ digits of p, then we can efficiently factor N.

  An adversary might know many digits of p if, for instance, we make the mistake of generating the random prime p by considering candidates of the form $2^{1023} + k$ for small (random) values of k (2^{1023} was chosen so that the resulting number has 1024 bits).

- Also, we must use a cryptographically secure PRG to generate p and q.

  If using a "bad" PRG or choosing seeds with too little entropy, then (especially among a large number of public keys generated this way) it becomes likely that (different) public keys N and N' share a prime factor p. In that case, everybody can determine $p = \gcd(N, N')$ and break both public keys.

  Indeed. For instance, in a study of Lenstra et. al., millions of public keys were collected and compared. Among the RSA moduli, about 0.2% shared a common prime factor with another one. That's terrible: if (different) public keys N and N' share a prime factor p, then everybody can determine $p = \gcd(N, N')$ and break both public keys.

  http://eprint.iacr.org/2012/064.pdf

- In that direction, is the security of public key cryptosystems like RSA in any way compromised when used by tens of millions of users?

  As noted above, millions of people using "bad" PRGs for generating RSA public keys make it likely that this weakness can be practically exploited.

  Similarly, for Diffie–Hellman and ElGamal, it is common to use fixed primes p. While fine in principle, this may be an issue if used by millions of users faced against an adversary Eve with vast resources. See, for instance: https://threatpost.com/prime-diffie-hellman-weakness-may-be-key-to-breaking-crypto/

Example 185. (side-channel attacks) For instance, by measuring the time it takes to decrypt messages as $m = c^d \pmod N$ in RSA, Eve might be able to reconstruct the secret key d (by choosing many ciphertexts c for Bob to decrypt).

This **timing attack**, first developed by Paul Kocher (1997), is particularly unsettling because it illustrates that the security of a system can be compromised even if mathematically everything is sound. This sort of attack is called a **side-channel attack**. It attacks the implementation (software and/or hardware) rather than the cryptographic algorithm.

See Section 6.2.3 in our book for more details on how d can be obtained in this attack.

In a similar spririt, there exist power attacks (measuring power instead of time during decryption) or fault attacks (for instance, injecting errors during computations):

https://en.wikipedia.org/wiki/Side-channel_attack

How to prevent? Implement RSA in such a way that no inferences can be drawn from the time and power consumption. One technique is to write **constant-time code**.

The specific attack on RSA can also be mitigated by **blinding**: instead of decrypting c, Bob decrypts cc_r where $c_r = r^e \pmod N$ with r being chosen at random. This results in $(cc_r)^d = mr \pmod N$ from which Bob determines m by multiplying with $r^{-1} \pmod N$. This takes away Eve's control over c.

Lesson. Do not implement crypto algorithms yourself!! Instead, use one of the well-tested open implementations.

It's kind of sad, isn't it? Don't come up with your own ciphers. Don't implement ciphers yourself...

But it is important to realize just how easy it is to implement these algorithms in such a way that security is compromised (even if the idea, intentions and algorithms are all sound and secure).