## Notes for Lecture 8

## 1 Chinese Remainder Theorem

Let $p \neq q$ be two primes. The Chinese Remainder Theorem (CRT) says that working modulo $n=p q$ is essentially the same as working modulo $p$ and modulo $q$ at the same time: more formally (for those comfortable with abstract algebra), that the ring $\mathbb{Z}_{n}$ is isomorphic to the product ring $\mathbb{Z}_{p} \times \mathbb{Z}_{q}$. (Actually, this is the "light" version of CRT, which is all we need for this course. The full-fledged version says that working modulo $a_{1} a_{2} \ldots a_{k}$, where $a_{i}$ are pairwise relatively prime, is the same as working simultaneously modulo $a_{1}, a_{2}, \ldots, a_{k}$.)

Here is an example. Consider all the values modulo 35 . They are in one-to-one correspondence with values modulo 5 and modulo 7 .

|  | 0 | $\underline{1}$ | 2 | $\underline{3}$ | 4 | 5 | 6 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0 | 0 | 15 | 30 | 10 | 25 | 5 | 20 |
| 1 | 21 | 1 | 16 | 31 | 11 | 26 | 6 |
| $\underline{2}$ | 7 | 22 | 2 | $\underline{17}$ | 32 | 12 | 27 |
| 3 | 28 | 8 | 23 | 3 | 18 | 33 | 13 |
| $\underline{4}$ | 14 | $\underline{29}$ | 9 | 24 | 4 | 19 | 34 |

Observe that if you want to add, say, 17 and 29 (underlined in the table), is the same as adding 3 (which is $17 \bmod 7)$ and $1($ which is $29 \bmod 7)$ modulo 7 to get 4 ; adding $2($ which is $17 \bmod 5)$ and $4($ which is $29 \bmod 5)$ modulo 5 to get 1 ; and then looking up the value corresponding to coordinates 4 and 1 in the table to get 11 (in a box in the table). Thus, we can do addition coordinatewise. Same for multiplication.

We now formally state and prove the observations above, generalized to $p$ and $q$ instead of 5 and 7 .
Theorem 1. Let $p \neq q$ be primes, $n=p q$. For each $a \in \mathbb{Z}_{p}, b \in \mathbb{Z}_{q}$, there is unique $c, 0 \leq c<n$ such that $c \equiv a \quad(\bmod p)$ and $c \equiv b \quad(\bmod q)$.

Proof. Let $r=p^{-1} \bmod q$ and $s=q^{-1} \bmod p$. Let $c^{\prime}=r p b+s q a$. Then $c^{\prime} \equiv r p b+s q a \equiv r \cdot 0 \cdot b+1 \cdot a \equiv a$ $(\bmod p)$, and $c^{\prime} \equiv r p b+s q a \equiv 1 \cdot b+s \cdot 0 \cdot a \equiv b \quad(\bmod q)$. Let $c=c^{\prime} \bmod p q$. Then $p q \mid\left(c-c^{\prime}\right)$, so $p \mid\left(c-c^{\prime}\right)$, so $c \equiv c^{\prime} \quad(\bmod p)$. Similarly, $c \equiv c^{\prime} \quad(\bmod q)$. Hence, $c$ satisfies all the conditions: $0 \leq c<n$, and $c \equiv a$ $(\bmod p)\left(\right.$ because $\left.c \equiv c^{\prime} \equiv a(\bmod p)\right)$, and $c \equiv b \quad(\bmod q)\left(\right.$ because $\left.c \equiv c^{\prime} \equiv b \quad(\bmod q)\right)$. Thus, for every pair $(a, b)$ there is a $c$. There are $p q=n$ possible pairs, and $n$ possible values of $c$, so for each pair there must be exactly one value of $c$, so it's unique for each $(a, b)$.

Denote by $\operatorname{crt}(a, b)$ the unique value of $c$ given by the above theorem. Then $\operatorname{crt}(a, b)=c$ if an only if $(a, b)=(c \bmod p, c \bmod q)$. Let $c_{1}=\operatorname{crt}\left(a_{1}, b_{1}\right), c_{2}=\operatorname{crt}\left(a_{2}, b_{2}\right)$, and $c_{3}=c_{1}+c_{2} \bmod n$. Then $c_{3} \bmod p=$ $\left(c_{1}+c_{2}\right) \bmod p=\left(a_{1}+a_{2}\right) \bmod p$ (because $n$ divides $c_{3}-c_{1}-c_{2}$, and therefore so does $p$ ) and similarly $c_{3} \bmod q=\left(b_{1}+b_{2}\right) \bmod q$. Hence $c_{3}=\operatorname{crt}\left(a_{1}+a_{2}, b_{1}+b_{2}\right)$. Same for multiplication. Thus, we can look at addition and multiplication modulo $n$ coordinatewise: modulo $p$ and modulo $q$.

We will denote by $\mathbb{Z}_{n}^{*}$ the set of values in $\mathbb{Z}_{n}$ that are relatively prime to $n$. Note that the "coordinates" of $\mathbb{Z}_{n}^{*}$ are in $\mathbb{Z}_{p}^{*}$ and $\mathbb{Z}_{q}^{*}$, and that $\mathbb{Z}_{n}^{*}$ has $(p-1)(q-1)$ elements.

Note that the above proof is constructive: that is, $c$ is efficiently (and, in fact, quite easily) computable given $a$ and $b$. Thus, it is often more efficient to work modulo $p$ and $q$ separately and the reconstruct the value modulo $n$ when it is needed.

## 2 Squares and Square Roots

Let $p>2$ be a prime. Let $Q R_{p}$ denote the set of squares in $\mathbb{Z}_{p}^{*}$. Recall from HW2 that for $a \in \mathbb{Z}_{p}^{*}$, if $a \in Q R_{p}$, then $a^{(p-1) / 2} \equiv 1$, and if $a \notin Q R_{p}$, then $a^{(p-1) / 2} \equiv-1$.

Suppose $p \equiv 3(\bmod 4)$. Take $s \in \mathbb{Z}_{p}^{*}$. It has two roots: $r$ and $-r$. Exactly one of these two roots is itself in $Q R_{p}$. Indeed, consider $r^{(p-1) / 2}$ and $(-r)^{(p-1) / 2}$. Since $(p-1) / 2$ is odd (because $p=4 k+3$ for some $k),(-r)^{(p-1) / 2}=-\left(r^{(p-1) / 2)}\right)$, so one is 1 and the other is -1 .

Hence, if we let $f_{p}(x): Q R_{p} \rightarrow Q R_{p}$ be the map $x \mapsto x^{2} \bmod p$, we see that for each $s \in Q R_{p}$, there exists a unique inverse $r \in Q R_{p}$ such that $f(r)=s$ (namely, $r$ is the square root of $s$ that is itself a square). So $f_{p}$ of $x$ is a permutation of $Q R_{p}$. Note that $f_{p}$ is easy to compute (just squaring) and easy to invert (as shown on HW2, it's easy to compute square roots modulo $p$ ).

Now let $p \neq q$ be two distinct odd primes, and let $n=p q$. Let $Q R_{n}$ denote the set of squares in $\mathbb{Z}_{n}^{*}$. Then if $s$ is a square modulo $n$, it is also a square modulo $p$ and $q$. Since it has two roots $\pm r_{1}$ modulo $p$ and two roots $\pm r_{2}$ modulo $q$, it has four roots modulo $n$ : $\operatorname{crt}\left( \pm r_{1}, \pm r_{2}\right)$.

Suppose both $p$ and $q$ are congruent to 3 modulo 4. Then exactly one of $\pm r_{1}$ is a square modulo $p$, and exactly one of $\pm r_{2}$ is a square modulo $q$, so exactly one of $\operatorname{crt}\left( \pm r_{1}, \pm r_{2}\right)$ is a square modulo $n$. Hence, if we let $f_{n}(x): Q R_{n} \rightarrow Q R_{n}$ be the map $x \mapsto x^{2} \bmod n$, we see that $f_{n}(x)$ is a permutation over $Q R_{n}$. Note that $f_{n}(x)$ is easy to compute. We will argue below that it is hard to invert-as hard as it is to factor $n$.

## 3 Blum-Blum-Shub Generator

The following construction is due to $[\mathrm{BBS} 86]^{1}$. Starting with a sufficiently long random seed, select two $k$-bit random primes $p, q$ that are 3 modulo 4 , let $n=p q$, and let $x$ be random element of $Q R_{n}$ (just select a random element of $\mathbb{Z}_{n}$, check if it's relatively prime with $n$, and square it). Let $x_{1}=x, x_{2}=f_{n}(x), x_{3}=$ $f_{n}\left(x_{2}\right), \ldots, x_{l}=f_{n}\left(x_{l-1}\right)$. Output the least significant bit for each $x_{i}$.

Note that this looks very much like the Blum-Micali generator, with exponentiation mod $p$ replaced with squaring $\bmod n$, and $B$ replaced with least significant bit. The proof is very similar, too. We simply need three facts: that the function $f_{n}$ is a permutation (already shown above), that computing $x$ from $x^{2} \bmod n$ is hard (discussed in the next section), and that computing the least significant bit of $x$ from $x^{2} \bmod n$ is as hard as computing all of $x$ (shown in [ACGS88]; an alternative proof is given is in [AGS03]; we will not discuss either here). These three facts correspond, in the Blum-Micali case, to the fact that modular exponentiation is a permutation of $\mathbb{Z}_{p}^{*}$ (which is used in the reduction because we have to know that the permutation has a unique inverse in order to show that the bits the reduction feeds to the adversary correspond to bits a generator would have generated), to the assumption that discrete logarithm is hard, and the theorem that $B(x)$ is as hard as to compute from $g^{x} \bmod p$ as $x$ itself.

This generator is more efficient than Blum-Micali: requires only one modular squaring per bit, instead of one one modular exponentiation. It is also based on a different (depending on whom you ask, more or less plausible) assumption: that factoring $n$ is hard. We will show this in the next section.

## 4 Square Roots Modulo a Composite are as Hard as Factoring

We want to justify why we believe it's hard to compute $x$ from $x^{2}$ modulo $n$. Indeed, let $s=r^{2} \bmod n$. Then $s$ has four square roots, as discussed above $\operatorname{crt}\left(r_{1}, r_{2}\right), \operatorname{crt}\left(-r_{1},-r_{2}\right), \operatorname{crt}\left(r_{1},-r_{2}\right), \operatorname{crt}\left(-r_{1}, r_{2}\right)$. Take two of these that are not negatives of each other, e.g., $r=\operatorname{crt}\left(r_{1}, r_{2}\right)$ and $r^{\prime}=\operatorname{crt}\left(r_{1},-r_{2}\right)$. Add them to get $r+r^{\prime}=\operatorname{crt}\left(2 r_{1}, 0\right)$. Thus, $r+r^{\prime} \equiv 0(\bmod q)$, so $q \mid\left(r+r^{\prime}\right)$. Note also that $r+r^{\prime} \not \equiv 0(\bmod p)$,

[^0]so $p X\left(r+r^{\prime}\right)$. Hence, $\operatorname{gcd}\left(r+r^{\prime}, n\right)=q$. Thus, if you know two such roots, you can factor $n$, by simply computing the greatest common divisor (this can be done quickly with Euclid's algorithm).

Now suppose we have an algorithm $A$ that computes square roots modulo $n$. We will use it to factor $n$ as follows: take a random $r \in \mathbb{Z}_{n}^{*}$, compute $s=r^{2} \bmod n$, and give $s$ to $A$. $A$ will return some root $r^{\prime}$ of $s$. Because $s$ has four roots and $r$ was chosen at random (and not given to $A$ ), no matter how $A$ works, $\operatorname{Pr}\left[r= \pm r^{\prime}\right]=1 / 2$. Hence, in half the cases, $\operatorname{gcd}\left(r+r^{\prime}, n\right)$ will give you a factor $p$ or $q$ of $n$.

Thus, we just proved (by contradiction and reduction, as usual) that if factoring $n$ is hard, so is computing square roots modulo $n$. Hence, the Blum-Blum-Shub generator is secure based on the following assumption:

Assumption 1. For any poly-time algorithm $F$, there exists a negligible function $\eta$ such that, if you generate random $k$-bit primes $p$ and $q$ that are both 3 modulo 4 , and let $n=p q, \operatorname{Pr}[F(n)=p] \leq \eta(k)$.

## References

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. SIAM Journal on Computing, 17(2):194-209, April 1988.
[AGS03] Adi Akavia, Shafi Goldwasser, and Muli Safra. Proving hardcore predicates using list decoding. In 44 th Annual Symposium on Foundations of Computer Science, Cambridge, Massachusetts, October 2003. IEEE.
[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. SIAM Journal on Computing, 15(2):364-383, May 1986.


[^0]:    ${ }^{1}$ Conference version published in Crypto in 1982.

