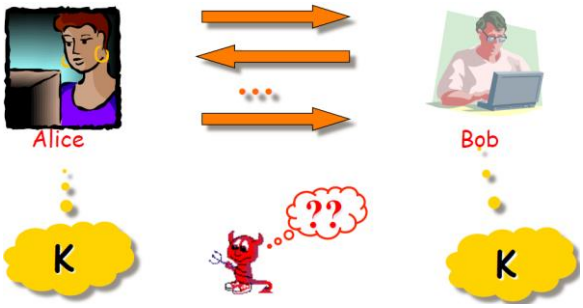


L'accordo su chiavi si basa sul seguente principio:

Alice e Bob, che non si sono mai visti in precedenza e di conseguenza non condividono nulla, iniziano un'interazione tra di loro, e alla fine dell'interazione, si accordano su una chiave simmetrica condivisa tra i due. Da questo deriva il concetto di accordo su una chiave, in maniera tale che l'attaccante che riesce ad origliare la conversazione che c'è tra Alice e Bob, non riesce a sapere il valore della chiave.

Quindi si vuole un accordo su una chiave che sia sicura in relazione a qualcuno che origlia la conversazione e un protocollo che resista anche se l'attaccante si mette in mezzo e cerca di imbrogliare Alice e Bob. Per ora ragioniamo solo sul fatto che l'attaccante possa origliare la conversazione tra Alice e Bob.

Il protocollo che viene usato maggiormente per l'accordo su chiavi è quello di **Diffie-Hellman** la cui diffusione è seconda solo ad RSA.

PROTOCOLLO DIFFIE-HELLMAN

Alice e Bob, dapprima condividono un primo p, e un generatore g di Z_p^* e questi 2 valori sono pubblici per tutti. Potrebbero addirittura essere gli stessi di quelli che vengono usati da un gruppo di persone.

Che cosa significa generatore? Se ad esempio scelgo il numero primo 11, voglio trovare un generatore di Z_{11}^* che è un numero che se lo vado ad elevare successivamente a tutte le potenze ottengo tutto Z_{11}^* come si vede nella figura al lato scegliendo 2:

$$\begin{cases} 2^{10} = 1024 = 1 \bmod 11 \\ 2^1 = 2 \bmod 11 \\ 2^8 = 256 = 3 \bmod 11 \\ 2^2 = 4 \bmod 11 \\ 2^4 = 16 = 5 \bmod 11 \\ 2^9 = 512 = 6 \bmod 11 \\ 2^7 = 128 = 7 \bmod 11 \\ 2^3 = 8 \bmod 11 \\ 2^6 = 64 = 9 \bmod 11 \\ 2^5 = 32 = 10 \bmod 11 \end{cases}$$

Se prendo il 2 e lo elevo a tutti i possibili esponenti, ottengo tutti i numeri da 1 a 10, cioè tutti i numeri di Z_{11}^*. Quindi g=2 è un generatore. In generale:

g è generatore di Z_p^* se $\{g^i \mid 1 \leq i \leq p-1\} = Z_p^*$

Esiste un risultato che ci dice che Z_n^* ha un generatore se e solo se n= $2,4,p^k,2p^k$ con p primo e $k \geq 1$. Poiché a noi p è sempre primo ciò implica che Z_p^* ha sempre un generatore.

Introduciamo un concetto di ordine di $\alpha \in Z_n^*$= il più piccolo intero positivo r tale che $\alpha^r = 1 \bmod n$

Ad esempio l'ordine di 4 è 9, l'ordine di 7 è 3 e così via, guardando ovviamente la tabella sotto.

L'ordine di 2 è 18. Inoltre, tutti gli ordini degli elementi devono essere dei divisori di 18.

Osserviamo Z_{19}^*:

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Preso ogni valore da 1 a 18 lo elevo ad ogni potenza fino ad arrivare a 18. Da questa tabella, si osserva che, se ho come numero primo 19, ci sono più di un generatore di Z_{19}^*, e sono 2 (perché elevando 2 ad ogni potenza ottengo tutti i numeri da 1 a 18), 3, 10,13,14,15. A noi ne basta solo 1.

Una volta ottenuto il generatore e il numero primo, Alice sceglie una $x \in Z_p$, Bob sceglie una $y \in Z_p$ a caso. Alice calcola g^x mod p e lo manda a Bob, Bob manda g^y mod p ad Alice. In questo modo hanno determinato la chiave. In quanto la chiave $K= g^{xy}$ mod p.

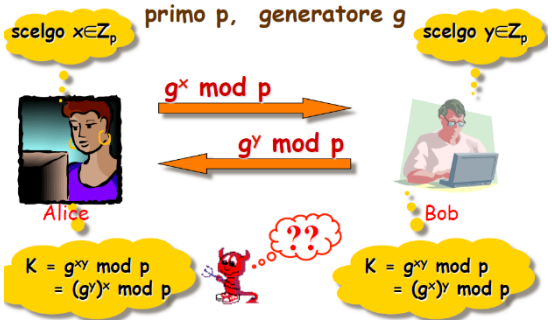
Alice per calcolarsela, fa $(g^y)^x$ mod p, perché riceve g^y e x l'ha scelta lei.

Bob per calcolarsela, fa $(g^x)^y$ mod p, perché riceve g^x e y l'ha scelto lui.

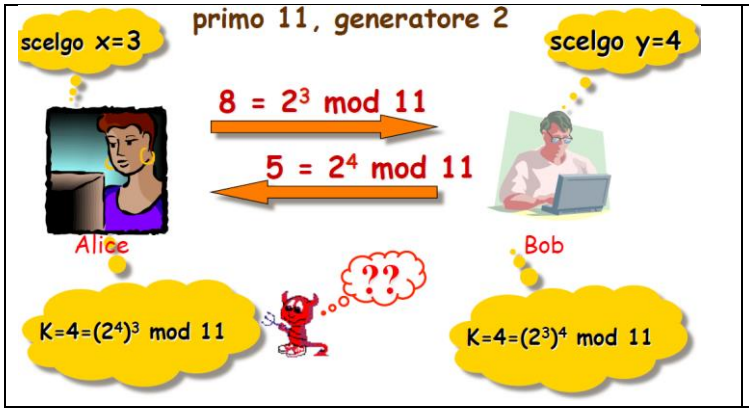
Sia Alice che Bob hanno calcolato la chiave K condivisa.

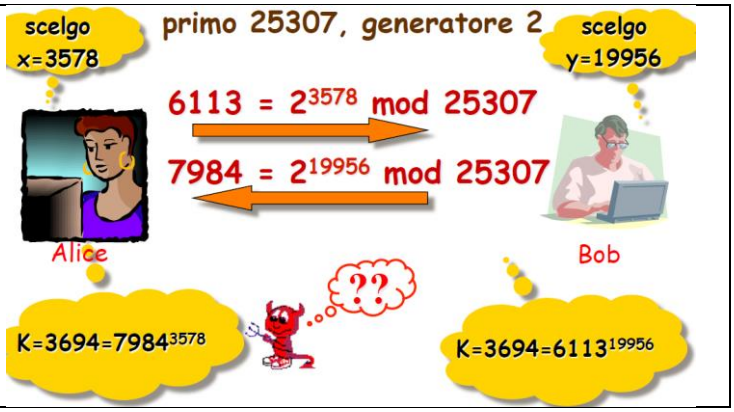
L'attaccante potrà vedere g^x e g^y ma il suo problema è calcolare g^{xy}.

Una volta calcolata questa chiave possono essere effettuate tutte le operazioni che desiderano, come utilizzarla come chiave simmetrica per un AES e per cifrare i messaggi.



Mostriamo 2 esempi pratici:





Essenzialmente il problema difficile su cui si basa lo scambio di chiavi di Diffie-Hellman non è la fattorizzazione in quanto non ci sono numeri composti, ma si basa sul problema del calcolo del logaritmo discreto, che significa, che il logaritmo di un numero è l'esponente che devo dare ad un altro numero per ottenere un valore noto. In generale:

Dati a,n,b, calcolare x tale che $a^x = b$ mod n, questo si chiama logaritmo discreto, perché lavoriamo in un'aritmetica in Z_n^*.

Ad esempio, dato $3^x = 7$ mod 13, la soluzione è x=6.

Questo è un problema difficile che si trova in altri crittosistemi come le firme digitali DSS. Se riuscissi a risolverlo, l'attaccante potrebbe rompere facilmente il protocollo, andando a calcolare le x e le y scelte da Alice e Bob.

Ci sono diversi algoritmi che risolvono il logaritmo discreto, e i migliori algoritmi che lo risolvono hanno una complessità che è simile a quella della fattorizzazione, nonostante non siamo legati. Questo dà un'idea del perché Diffie-Hellman continua ad esistere.

Il vero problema su cui si basa l'accordo di chiavi è il seguente:

Dati in input un primo p, generatore g, g^x mod p, g^y mod p, calcolare g^{xy} mod p.

Il miglior algoritmo conosciuto calcola prima il logaritmo discreto di g^x o g^y e poi calcola il valore finale.

Rompere Diffie-Hellman non è chiaro se sia equivalente a estrarre logaritmi discreti.

Come vengono scelti p e g? Il primo algoritmo immediato che posso usare è il seguente: prima scelgo p e poi scelgo il generatore:

```
Scegli_Generatore_Naive (p)
1. Scegli a caso g in Z_p^*
2. If {g^i | 1≤i≤ p-1} = Z_p^*
        then trovato
        else goto 1.
```

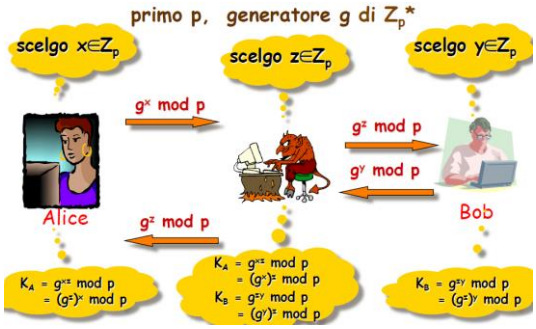
Il problema è, come faccio a sapere se un valore calcolato è un generatore o meno? L'unico algoritmo efficiente necessita dei fattori primi di p-1, perché deve verificare la proprietà dell'ordine degli elementi. Se non ho la fattorizzazione di p-1 non riesco a verificare se il valore è un generatore o meno. Fare una cosa del genere risulta essere impossibile in quanto fattorizzare un numero di 2048 bit è molto complesso da fare. Serve un metodo diverso, che si può dire essere un algoritmo inverso nel senso che genera p in maniera tale che si conosce la fattorizzazione di p-1:

1. Scegli a caso 2 numeri primi p_1 p_2
2. $p \leftarrow 1 + 2p_1p_2$
3. Se p non è primo, go to 1.
4. $g \leftarrow$ **Scegli_generatore**$(p,(2,1,p_1,1,p_2,1))$

Scelgo due numeri primi p_1 e p_2, vado a calcolare un numero p che è uguale a $1 + 2p_1p_2$. Testo se è primo, se non lo è itero questo step, altrimenti conosco p, conosco la fattorizzazione di p-1 che è $2p_1p_2$ e posso generare un generatore. Questo è l'algoritmo che viene utilizzato.

Il protocollo Diffie-Hellman è sicuro contro attaccanti passivi in quanto, può semplicemente origliare.

Non è sicuro contro attacchi man-in-the-middle, cioè che l'attaccante si mette nella comunicazione tra Alice e Bob e parla a uno dei 2 come se fosse l'altro. Quindi Alice e Bob pensano di parlare tra di loro, in realtà parlano con un terzo. Il principio è il seguente:



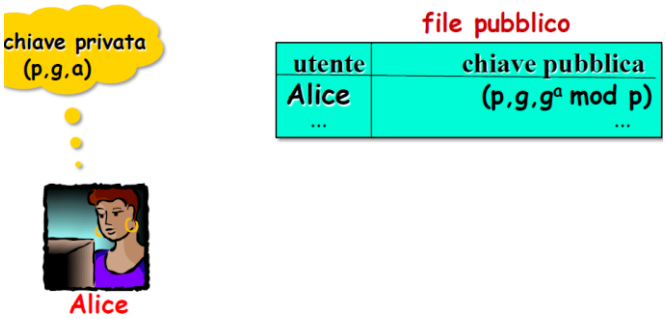
Dal momento che Alice e Bob non condividono niente, Alice sceglie x e manda $g^x \bmod p$, questo g^x non arriva a Bob ma viene intercettato dall'attaccante che non lo inoltra a Bob e lo tiene per se. A Bob invia un g^z in cui z viene calcolato dall'attaccante stesso. Bob riceve un valore a caso, pensando che gli è arrivato da Alice. A questo punto sceglie y e invia g^y ad Alice, ma viene intercettato dall'attaccante che lo tiene per se e ad Alice manda invece uno g^z dove z è il valore scelto dall'attaccante. L'idea quindi è di fermare i messaggi.

Alice quindi invia x e riceve z, quindi la sua chiave è g^{zx}, la chiave che Bob calcola è data da y e z e quindi g^{zy}. L'attaccante è in grado di calcolare entrambe queste chiavi. Quindi Alice e Bob non hanno creato una chiave in comune con l'altro, bensì con l'attaccante. Il messaggio che manderanno Alice e Bob verranno intercettati dall'attaccante e potrà decifrarli.

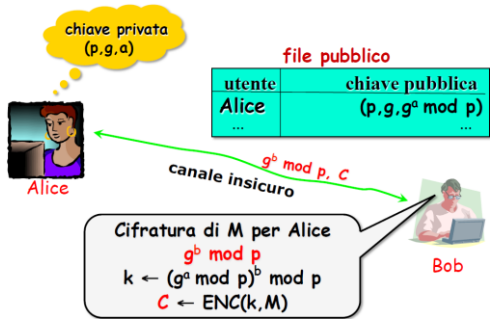
Il vero motivo per cui questo accordo non è più sicuro è perché non c'è autenticazione.

Possiamo usare Diffie-Hellman come cifrario asimmetrico? Devo prima definirmi una chiave pubblica, quella privata, come funziona la cifratura e come funziona la decifratura.

Alice mette g^x come chiave pubblica e x come chiave privata:



Alice mette p e g come valori pubblici ma quelli tutti li conoscono dall'inizio, e mette $g^a \bmod p$ dove a è la sua chiave privata. In questo modo, quando Bob deve cifrare qualcosa per Alice, in qualche modo, il messaggio che deve ricevere da Alice, anziché riceverlo lo va a leggere nella chiave pubblica e a questo punto, per accordarsi su una chiave, basta semplicemente che manda il suo messaggio. Quindi se deve cifrare:



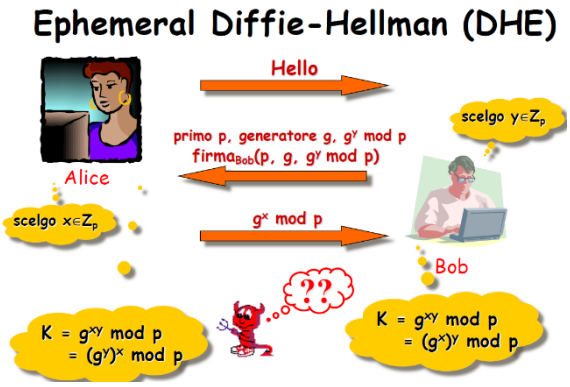
Sceglie un valore casuale b, manda ad Alice $g^b \bmod p$, in questo modo si sono accordati su una chiave simmetrica, la chiave k che sarebbe $(g^a \bmod p)^b$. Questa chiave k è una chiave di sessione e quindi se deve cifrare qualcosa utilizza questa chiave k appena calcolata, per cifrare il messaggio M.

L'accordo di una chiave diventa lo stabilire una chiave di sessione che viene poi utilizzata per cifrare il messaggio. In questo modo, il problema dell'autenticazione è risolto, in quanto essendoci un file pubblico mi posso fidare di esso.

RSA e Diffie-Hellman sono due modi diversi di risolvere lo stesso problema che è quello della crittografia a chiave pubblica e sull'accordo di chiavi.

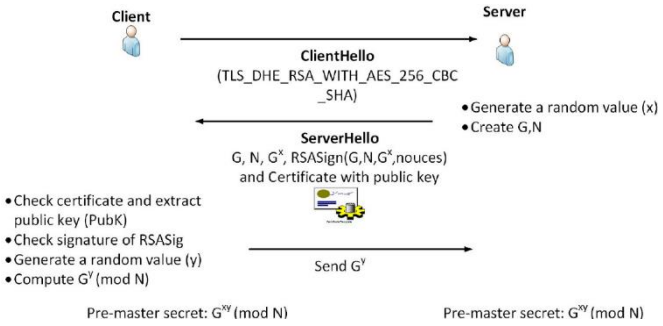
Alcuni protocolli si basano sulla modalità **forward secrecy**, che si usa in crittografia a chiave pubblica. Se qualcuno viene a conoscenza della chiave privata, allora riesce a rompere i messaggi futuri che verranno cifrati e decifrati con la chiave privata, ma non i messaggi passati. Questa caratteristica è presente in alcuni protocolli. Naturalmente questa proprietà vale se si effettua qualche modifica all'algoritmo. L'idea essenzialmente è di utilizzare un accordo di chiavi Diffie-Hellman che deve essere autenticato in cui la chiave privata cambia sempre. Per esempio, questo accordo di chiavi potrebbe avvenire in questo modo:

Alice manda un messaggio “Hello”. Bob gli invia il primo *p*, il generatore *g*, g^y mod p, dove *y* è una chiave privata scelta a caso dopo aver ricevuto il primo messaggio. Questa tripletta viene firmata con la chiave di Bob: ***firma$_{Bob}$(p, g, g^y mod p)***. Una volta che Alice riceve questa firma, sceglie una *x*, invia g^x mod p a Bob, in modo da condividere la chiave di sessione dell’accordo di Diffie-Hellman K= g^{xy} mod p. In questo modo, la conoscenza della chiave attuale, che è una chiave di firma, non serve a niente per decifrare i messaggi passati. Questa è l’idea della forward secrecy. Nel momento in cui si creerà una nuova sessione tra Alice e Bob, verrà generata una nuova chiave di sessione, tenendo conto che Bob non avrà bisogno di rigenerare *p* e *g*, ma dovrà calcolare solamente un nuovo *y*. Se avessi usato RSA, avrei dovuto generare n da capo e avrebbe richiesto molto tempo.



Questa idea viene usata in TLS 1.3:

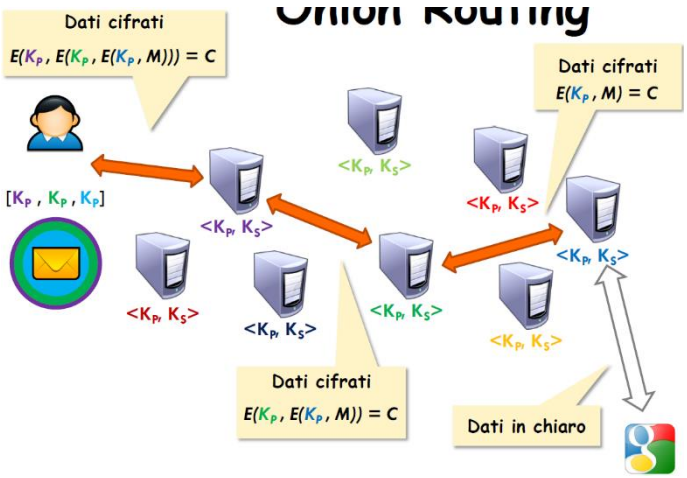
Se si vuole usare questo servizio di forward secrecy, è il Client a deciderlo, mandando il messaggio di Hello in cui essenzialmente manda la suite crittografica che lui intende utilizzare (DHE indica l’utilizzo di Diffie-Hellman con utilizzo di forward secrecy). Il Server, ricevuta la decisione del Client, genera x, crea G e N che sarebbero il generatore e il numero primo e invia al Client con RSA la firma di G,N,G^x e il certificato con la chiave pubblica. Il Client, una volta ricevuto questo messaggio, esegue degli step di verifica e genera un numero casuale y e manda al Server G^y. In questo modo Client e Server posseggono la chiave di sessione calcolata come: G^{XY} (mod N). Questa chiave viene chiamata Pre-master secret (verrà spiegato nelle prossime lezioni il perché), che rappresenta una gerarchia di chiavi.



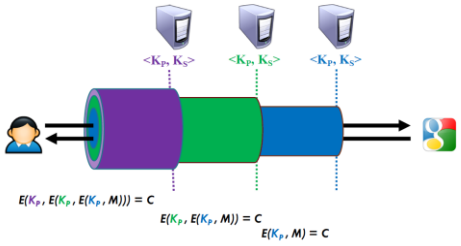
Questa idea viene usata anche in TOR (The Onion Router):

l’idea è di costruire un’azione (lettura di un sito web, etc..), mediante un cammino scelto di volta in volta con un layer di cifrature in maniera tale da evitare che i nodi che si trovano su questo cammino possano conoscere informazioni riservate. L’idea di Onion Routing è la seguente:

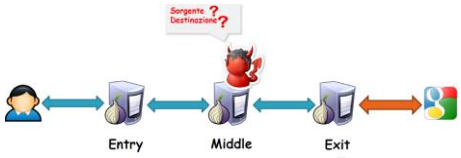
L’utente, posto in alto a sinistra, vuole visitare Google in anonimia. C’è una serie di nodi di cui l’utente possiede l’indirizzo, e ne sceglie 3 fissati (il cammino con le frecce arancioni). Scegliendo i nodi in questo cammino passano tutte informazioni cifrate, solo dall’ultimo nodo al sito web il dato è in chiaro poiché si accede ad un sito. Una volta mandata la richiesta al sito, quest’ultimo genera una risposta che sarà mandata nello stesso cammino in cui il client ha mandato la richiesta (infatti il cammino è formato da archi con frecce a doppia direzione). In generale, l’interazione tra utente e sito avviene sul cammino stabilito all’inizio dal client. Poiché è stato esplicitato che le informazioni inviate dal client vengono cifrate, all’interno dei nodi dentro cui passano le informazioni non si potrà mai determinare l’informazione in chiaro. Il singolo nodo conosce solo 2 informazioni: chi viene prima di esso nel cammino, e chi è il nodo successivo. Chiaramente l’operazione di cifratura deve avvenire in maniera tale che se un attaccante in un tempo qualsiasi cattura i nodi, anche avendo catturato il traffico in precedenza non deve riuscire ad ottenere informazioni chiare sulla transazione che è avvenuta tra Client e sito web.



Se ad esempio il Client deve mandare un messaggio, il Client cifra un messaggio con la chiave del nodo successivo ed una volta mandato al nodo successivo, quest’ultimo, cifrerà il messaggio con la chiave del nodo successivo e così via… L’idea di questo Onion Routing è che ci sono una serie di nodi scelti a caso dall’utente, e tutte le informazioni che hanno tutti i nodi sono locali a chi viene prima e a chi viene dopo, quindi non hanno una visione complessiva dell’azione che si deve eseguire.



Si giustifica quindi il concetto di Onion (cipolla) routing: i messaggi che vengono inviati sono cifrati più volte.



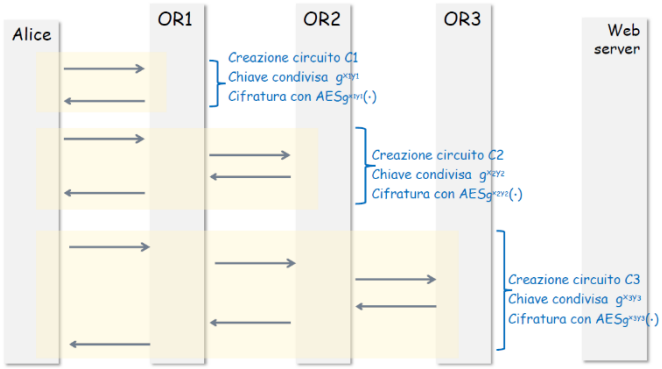
Un attaccante, in base al nodo in cui si posiziona:

- Se decide di posizionarsi in **entry** conosce solo la sorgente.
- Se decide di posizionarsi in **middle** non conosce nulla riguardo la sorgente e la destinazione.
- Se decide di posizionarsi in **exit** conosce solo informazioni riguardanti la destinazione.

Tutte le informazioni spiegate fino ad ora sono servite per dare l'idea di come funziona la Forward Secrecy in TOR, che viene illustrata ora:

A sinistra è collocata Alice che vuole ottenere un'informazione (visitare un sito web). Sulla destra è posto il Web server che fornisce l'informazione. Al centro fra Alice e il Web Server ci sono: OR1 (nodo d'entrata), OR2 (nodo centrale), OR3 (nodo di uscita). Si ricorda che questi 3 nodi vengono scelti da Alice. Essenzialmente in TOR ci sono 3 macrostadi:

Alice costruisce un circuito con il primo nodo (OR1): si accorda su una chiave con il primo nodo (con caratteristica di Forward Secrecy). Avviene uno scambio di informazioni e alla fine viene trovata una chiave di sessione condivisa g^{x1y1} che è una chiave Diffie-Hellman. Nel momento in cui si è stabilito questo circuito tutte le successive comunicazioni che avvengono tra Alice e il primo nodo utilizzano questa chiave privata che è stata generata in comune accordo. Per garantire sicurezza, la comunicazione tra Alice e OR1 avviene mediante *AES* sulla chiave g^{x1y1}.

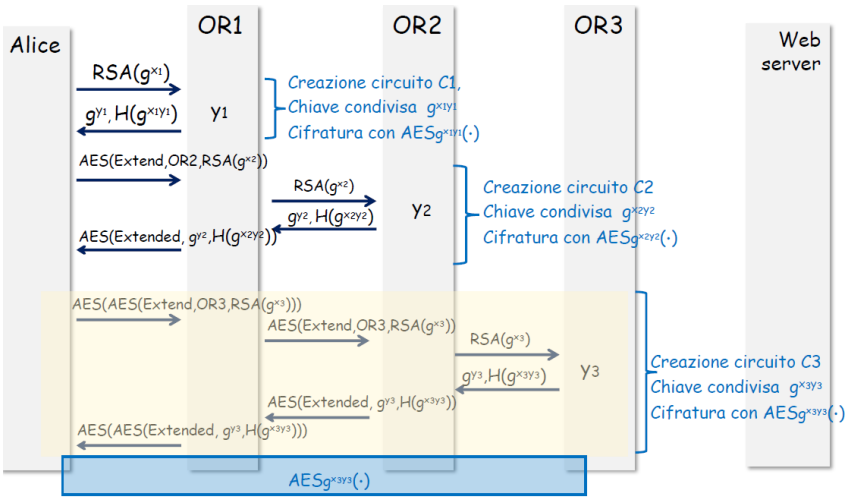


Dopo aver fissato il primo nodo, Alice dialoga con il secondo nodo mediante il primo nodo con cui ha già stabilito un collegamento sicuro: come si vede dall'immagine, Alice comunica con il primo nodo, che a sua volta comunica con il secondo nodo, per stabilire una chiave di sessione tra Alice e OR2 mediante OR1. Naturalmente la chiave che viene generata g^{x2y2} è conosciuta solo da Alice e OR2. Tutte le conversazioni che avvengono tra Alice e OR2 avvengono mediante AES sulla chiave g^{x2y2}

Su questa idea, si crea anche la comunicazione tra Alice e OR3, i quali comunicheranno mediante AES sulla chiave g^{x3y3}.

Tutte le informazioni mandate e ricevute tra Alice e il Web server passano su questo circuito di volta in volta. C'è anonimia perché Alice cambia in continuazione OR1, OR2 e OR3.

Questi concetti vengono messi in pratica in questo modo:

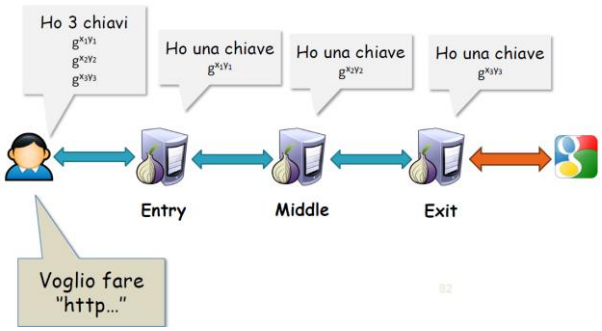


Alice deve stabilire la chiave di sessione con OR1. Per fare ciò cifra (utilizzando la chiave pubblica di OR1) con RSA il suo g^{x1} al primo nodo. OR1 genera il suo y1 e invia ad Alice g^{y1} e un Hash della chiave condivisa, H(g^{x1y1}) che serve per verificare l'integrità della chiave condivisa. In questo modo, Alice e OR1 hanno creato il circuito con chiave condivisa g^{x1y1}. Ogni volta che ci sarà una conversazione tra Alice e il primo nodo, avverrà mediante AES sulla chiave g^{x1y1}. Se un attaccante riuscisse ad ottenere informazioni sul primo nodo, conoscendo dunque y1, non potrà mai sapere la chiave g^{x1y1} perché viene stabilita di volta in volta (questa è la giustificazione sull'utilizzo della Forward Secrecy).

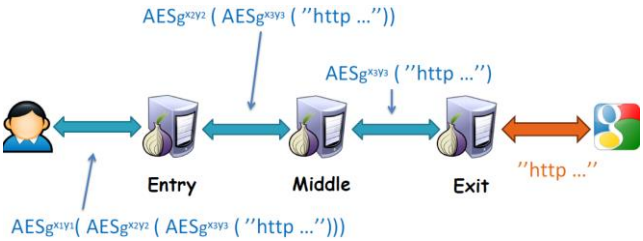
Adesso Alice deve comunicare con il secondo nodo e deve quindi estendere il circuito. L'idea è la seguente: ***AES(Extend,OR2,RSA(g^{x2}))***. **AES** indica la comunicazione che avviene tra Alice e il primo nodo. **Extend** indica che Alice vuole estendere il circuito con **OR2** a cui manda l'informazione **RSA(g^{x2})**, dove x2 è un nuovo valore casuale scelto da Alice. RSA(g^{x2}) avviene mediante la chiave pubblica di OR2. Il nodo OR2 decifra il contenuto e ottiene y2 e invia ad OR1 g^{y2} e l'Hash della chiave condivisa, H^{gx2y2}. OR1 manda in maniera cifrata (con AES) queste informazioni ad Alice. Viene calcolata la chiave di sessione tra Alice e OR2.

Sulla stessa idea viene generato il circuito tra Alice e OR3.

Dopo tutto questo processo di creazione di circuiti, la conclusione è la seguente:



Alice possiede 3 chiavi di sessione, una per ogni nodo. Ogni qualvolta Alice vuole visitare un sito:



Alice invia un'informazione che è cifrata utilizzando tutte le chiavi che ha stabilito nella creazione dei circuiti di volta in volta. L'idea è la seguente: cifra con la prima chiave qualcosa che è cifrato con la seconda chiave, che a sua volta è cifrato con la terza chiave per generare poi il valore reale che viene inoltrato.

Il protocollo Diffie-Hellman viene frequentemente usato per ottenere Forward Secrecy per la velocità della generazione della chiave.