Skype(v2.5) Protocol Analysis (French)

Author : Ouanilo MEDEGAN

http://www.oklabs.net

ARCHITECTURE & ENTITES DU RESEAU

Le réseau skype étant un réseau Peer 2 Peer, chaque client se comporte à la fois comme un client et un serveur. Chaque client skype est un « nœud » du réseau. Cependant, sous réserves de propriétés qualifiantes (bande passante, puissance CPU, uptime, configuration réseau), tout nœud est susceptible d'être élu au rang de « Super-Nœud ». Un Super-Nœud est un nœud qui a pour but de servir « d'informateur » aux nœuds classiques ou de relayer les communications des nœuds se trouvant dans des configurations réseau spécial.

Chaque nœud est représenté par un NodeID unique (aléatoire et « taggué » avec des données environnementales uniques telles que le numéro de série du système d'exploitation, le numéro de série du disque dur, etc..). Le NodeID est composé de 8 octets (soit deux doubles mots qu'on désignera par NodeID.H et NodeID.L).

Les Super-Nœuds sont regroupés par « Slot », un slot étant un groupe de 8~10 Super-Nœuds. (Le nombre de slot est estimé à 2050, mais ce nombre est calculable).

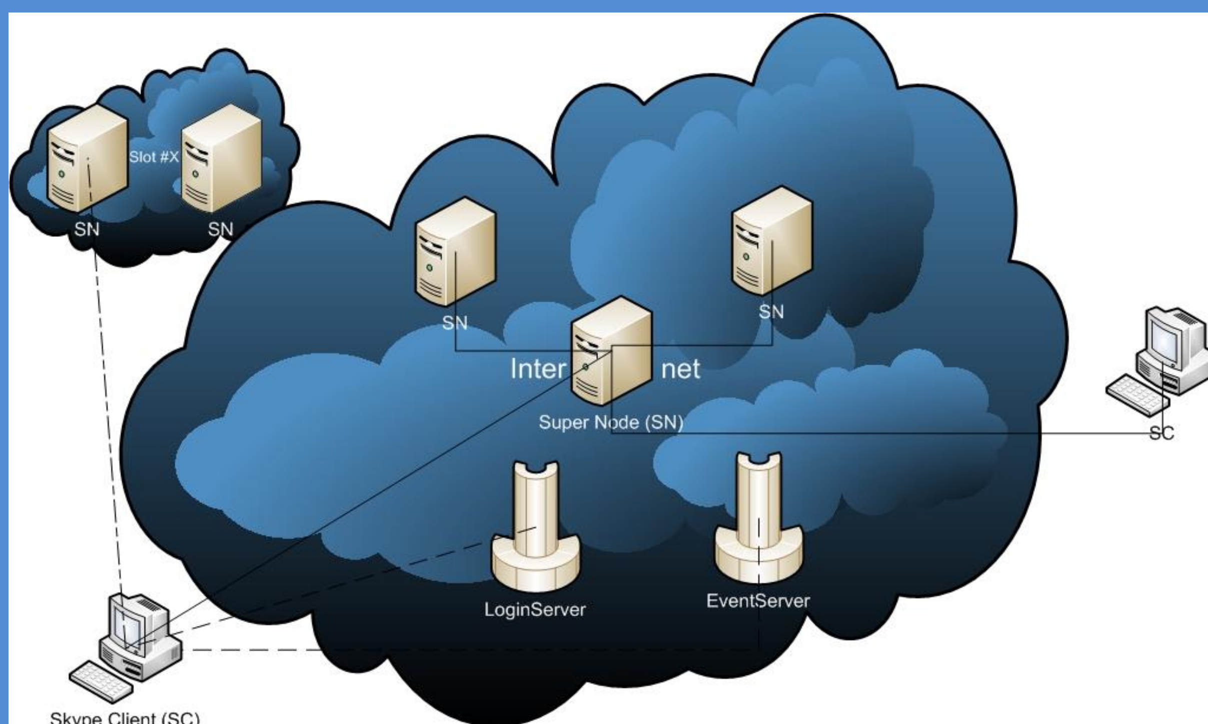
Notons que chaque binaire skype contient le code propre au comportement d'un nœud classique ou d'un super-nœud.

Il existe aussi des structures centrales « indispensables » au fonctionnement du réseau, à savoir principalement :

- LoginServer : Ceux-ci ont pour but de valider les accès d'un client lors de son authentification, afin de lui permettre d'être reconnu authentique sur le réseau.

- EventServer : Ceux-ci ont un rôle de « cache » d'information pour chaque client authentifié sur le réseau. Ils stockent les listes de contacts, les historiques de chat etc..

« Skype Technologies S.A. » a publié un document [1] décrivant globalement l'architecture du réseau et quelques mécanismes de celui-ci.



Sur ce schéma, on peut voir les relations d'un nœud skype classique avec les autres nœuds, les architectures centrales, les super-nœuds et les liens entre ces derniers.

Un Client Skype passe par plusieurs états au cours d'une session complète :

- Disconnected
- Registering
- Registered

[1] http://www.skype.com/security/guide-for-network-admins-30beta.pdf

- Initiliazing chat/voice/video session
- Session in Progress

Le protocole Skype est un protocole vaguement comparable au protocole RPC, dans la mesure où les requêtes sont spécifiés par des identifiants numérique, les paramètres des requêtes sont typés (unsigned int, string, inet_addr, object list, ..). (Voir plus bas pour plus de détail).

DESCRIPTION GLOBALE DU PASSAGE DE L'ETAT *DISCONNECTED* A *REGISTERED*

Dans notre scenario nous allons supposer que le client Skype en question en est à sa première connexion sur le réseau. Il ne connait donc aucune information cachée (adresse de super-nœuds, liste de contacts, etc..) .

Le client contient dans son code, une liste d'adresse de super-nœuds dits de « bootstrap » (voir annexe) qui lui permettront de se lancer dans la phase de Register.

- Scan de super-nœuds à la recherche d'un super-nœud pour manager la session du client

  Notons que tous les paquets UDP ou TCP, entrants comme sortants sont cryptés en utilisant la primitive de cryptage RC4 associée à une clé générée à partir d'un « seed » lui-même généré à partir de paramètres environnementaux tels que l'adresse IP du destinataire, l'adresse publique du client, vecteur d'initialisation, ID du paquet..

En réalité lors de l'envoi du tout premier paquet le client ne connait pas son adresse publique. Il va donc utiliser comme adresse publique pour générer le « seed » l'ip : » 0.0.0.0 ». Le SN qui recevra le paquet utilisera pour décrypter un seed générer à partir de la vraie IP publique du client qui envoie le paquet. La décryption échoue donc logiquement, le SN renvoie donc un paquet de type « NACK », contenant l'ip publique du client dont il a reçu le paquet (ainsi qu'un challenge). Celui met donc à jour son IP publique et renvoie le paquet contenant le challenge afin d'attester de son authenticité. Une fois cela fait on peut considérer que le scan proprement dit commence.

Le client va commencer par envoyer via UDP à tous les SN connus un « probe », auquel le SN va répondre soit par un « probe accept » ou par un « probe reject ».
Dans le cas d'un « probe reject », le SN va renvoyé avec sa réponse une liste de SN qu'il connait afin de fournir au client les moyens de continuer son scan sur ces SN là.
Dans le cas d'un « probe accept », le client va donc jusqu'à obtenir un canal TCP avec le SN, essayer de se connecter tour à tour sur le port en « listen » du SN, puis sur le port 443, puis sur le port 80. Si aucun de ces ports ne permet l' établissement d'une connexion TCP on continue la chaîne de probe jusqu'au prochain « probe accept ».
Notre environnement d'étude ne nous permettant pas la connexion TCP sur un port « custom » , on va considérer que le client se connecte par le port 443.
Une fois que la connexion TCP est établie avec le SN, le client va envoyer un paquet de type « HTTPS Handshake Request »,

auquel le SN va répondre par un paquet de type « HTTPS Handshake Reply ». Cet échange est surement destiné à obtenir la confiance des I(D|P)S.
Les échanges conséquents entre le client et le SN à partir de ce moment la vont être cryptés avec un flux RC4 pour les données sortantes et un autre flux RC4 pour les données entrantes.
Le client va envoyer au SN, un paquet contenant le « seed » à partir duquel il va générer la clé pour initialiser le flux RC4 pour les données sortantes. Le SN va lui répondre avec un paquet contenant le « seed » à partir duquel lui il va initialiser le flux RC4 pour les données qu'il enverra au client, donc les données entrantes du client.
Une fois les flux initialisés, le client va donc procéder a une demande « client accept » qui est donc sa demande présentée au SN de le manager durant se session.
Le SN répond soit par un « client refused », auquel cas la connexion est fermé et le scan de SN continue, ou par un « client accepted », auquel cas, le scan s'arrete car le client a donc a ce moment la trouvé le SN qui allait le manager durant la session. On va désigner ce SN par le terme de ParentNode (PN).
En général avec une réponse « client accepted » , le client reçoit des statistiques du réseau à savoir le nombre de clients connectés, ainsi qu'une liste de « channels » aux quels souscrire afin de recevoir plus d'informations à jour du réseau. Si le client a déjà auparavant (lors d'une connexion antérieure) caché les versions des channels proposés il n'y souscrira pas.

- Authentification auprès d'un LoginServer (LS)

Le client doit maintenant s'authentifier dans le but de recevoir des données signées par l'autorité de « Skype Technologies », données signées qui lui permettront par un système de clés privé/publique d'être reconnu comme authentique par les autres nœuds et SN du réseau.
Ici aussi le client contient dans son code, une liste d'adresse de LoginServer sur lesquelles il va tenter d'ouvrir une connexion tour à tour sur le port en « listen » du LS, puis sur le port 443 et sur le port 80, jusqu'à ce qu'il réussisse à établir une connexion. Nous allons ici aussi (contraintes de l'environnement d'étude) supposé que le client obtient une connexion sur le port 443.
Il va donc commencer par envoyer un paquet de type « HTTPS Mini HandShake Request » auquel le LS va répondre par un paquet de type « HTTPS Mini HandShake Reply ».
Une fois cet échange accompli (dans le but de savoir si le serveur est « up » et apte à traiter la requête), le client va donc envoyer au LS un paquet contenant des blocs de données cryptées contenant l'un un hash du login et du mot passe du compte pour lequel la session sera ouverte, l'autre que une clé publique (clé de 1024 bits générée à l'exécution et qui va servir tout le long de la session) qui va être signée par l'autorité « Skype Technologies » afin d'assure l'authenticité du compte.
Avec ces blocs cryptés le client envoie aussi dans le paquet certains paramètres moins importants tels que le « username », la version du client, l'adresse publique connue, le NodeID etc..
Voici le mécanisme de génération du paquet :

- Le client génère une clé privée « PV » et une clé publique « PB » qui vont servir tout le long de la session, chacune étant longue de 1024 bits.

- Le client génère aussi une clé de session aléatoire de 192 octets « SK ».
- Le client va générer suivant l'algorithme suivant un « buffer » « AesKey » de 32 octets :

  - Les 20 premiers octets de AesKey sont les 20 premiers octets du hash SHA1 de la chaîne "\x00\x00\x00\x00" concaténée à SK.
  - Les 12 derniers octets de AesKey sont le 12 premiers octets du hash SHA1 de la chaîne "\x00\x00\x00\x01" concaténée à SK.
- AesKey est utilisé comme clé pour initialiser un flux de cryptage AES256 en mode « counter ».
- Notons que le binaire skype contient un liste de clés publiques de différentes longueurs (3 clés de 1536, 9 de 2048, et 2 de 4096) (voir annexe).
  Le client va utiliser la deuxième clé de 1536 bits ('a8f223612f4f5fc81…') pour effectuer un « Public Encrypt » RSA (donc cryptage avec la clé publique) de SK. On a donc SK encrypté RSA(SK).
- Ce bloc RSA(SK) de 192 octets sera envoyé comme paramètre dans le paquet envoyé au LS.
- Le client va générer le hash MD5 de la chaine « login|"\nskyper\n"|password» où « login » est le login du compte et « password » le mot de passe.
- Ce hash md5 va être stocké dans une liste de paramètre avec les autres paramètres moins importants comme l'adresse IP publique, la version du client etc..

- A cette liste de paramètres va être rajoutée un « buffer » de 128 octets (1024 bits) contenant la clé publique de la session.
- La liste des paramètres en question va être crypté via le flux AES256 initialisé plus haut, et le résultat va etre rajouté au paquet en plus du bloc RSA(SK).

Une fois le paquet généré le client l'envoie au LS, qui suivants la validité du hash des accès (login et mot de passe) va soit renvoyé un paquet contenant un code décrivant un échec, soit un paquet contenant un code décrivant la réussite de l'authentification et aussi un bloc de données (désigné « SignedCredentials ») signé avec la clé privée de « Skype Technologies » associée a l'une des clés publiques contenues dans le binaire Skype.

Ce bloc SignedCredentials sera envoyé à tout pair ayant besoin d'être sur de notre authenticité. Au lieu que ce pair ait à vérifier auprès d'un serveur central si le compte utilisé par le client s'est au préalable authentifié avec succès, il n'a qu'a vérifier s'il arrive à décrypter les SignedCredentials reçues avec une des clés publiques connues (l'index de la clé à utiliser est spécifier en entête des SignedCredentials), le cas échéant, cela voudrait dire que les SignedCredentials ont bien été signés par un LS et donc que le compte s'est bien authentifié au préalable.

Les SignedCredentials contiennent une fois décryptés, la clé publique associé a la session, le login du compte associé à la session ainsi qu'en secondes la durée pendant laquelle ces SignedCredentials sont valides.

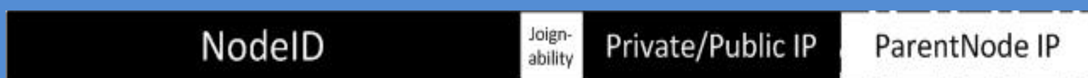
Une fois les SignedCredentials reçues et donc l'authentification menée à terme avec succès, le client se déconnecte du LS.

- Brodcast de la présence sur le réseau P2P

Notons tout d'abord qu'à chaque login est associé un slot d'activité. C'est-à-dire pour le login 'oj.med' par exemple le slot de super-nœuds dans lequel on se manifestera sera le slot #344. L'identifiant du slot auquel est associé un login se calcule suivant un algorithme où intervient un CRC32 sur les premiers caractères du login.
Le broadcast de la présence s'effectue seulement via des super-nœuds du slot associé au login, car tout essai de localisation du client par d'autres clients ayant le compte dans leur liste d'amis, sera adressée a des super-nœuds du slot associé au login.
Supposons que nous sommes 'oj.med'. Nous allons broadcasté notre présence via des super-nœuds du slot #344. Un client se connecte en utilisant un compte 'ami.de.oj.med' dont la liste de contacts contient 'oj.med' comme ami. Le client 'ami.de.oj.med' va donc calculer que le slot ou il doit adresser sa recherche est le slot #344 ou le client utilisant le compte 'oj.med' a dû broadcasté sa présence.

Comment s'effectue donc le brodcast de la présence ?
Afin d'être localisable sur le réseau on doit envoyer à environ au moins 3 super-nœuds du slot associé au login de la session, un paquet de type « DirBlob » (voir plus loin) ayant comme charge utile une liste d'objet encapsulant un buffer décrivant la localistaion du client et forgé comme suit :

- Les huits premiers octets du buffer représentent le NodeID du client. (NETWORKORDER(NodeID.L)|NETWORKORDER(NodeID.H))
- Le 9ème octet du buffer prend la valeur 0x01 pour spécifier que le client n'est joignable que via un super-nœud qui lui sert de ParentNode. Si le client est directement joignable cet octet prend la valeur 0x00 .
- Si le client est directement joignable les 4 octets suivant contiennent IP Publique du client, sinon ils contiennent son IP privée.
- Si et seulement si le client n'est pas joignable, le buffer va contenir 4 octets supplémentaires, contenants l'IP du ParentNode du client.



Une fois le paquet de type DirBlob contenant la localisation du client forgé celui-ci doit donc le diffusé. Le client va donc envoyé a son ParentNode de lui renvoyé les adresses IP d'au plus 12 SN appartenant au slot associé à son login. Une fois ces adresses connues, le client va en UDP envoyé a chacun de ces SN le paquet DirBlob encapsulé dans la liste d'objets en paramètres de la fonction associée au broadcast de la présence.

Dès lors tout autre client Skype désirant contact notre client, recevra, après avoir adressé à au plus 5 SN du slot associé au login de la session de notre client, une requête de recherche de localisation de notre client, en réponse, le paquet DirBlob que nous avons diffusé dans notre slot. Il pourra alors nous

contacter par le biais de notre ParentNode ou directement si on est publiquement joignable.

A partir de ce moment le client est joignable par ses contacts amis, et peut donc être considéré comme en l'état REGISTERED.

DESCRIPTION GLOBALE DE L'ETAT *REGISTERED*

En l'état REGISTERED, le client Skype a la possibilité de récupérer la liste de contacts associée au compte de la session. Depuis les versions 1.5 de Skype, la liste des contacts n'est plus stocké dans un fichier local propre à chaque PC mais est maintenue à jour et accessible de partout via les EventServers (ES).

Il est a noté que en l'état REGISTERED le client peut à tout moment recevoir des pings provenant d'autres clients utilisant des comptes présents dans sa liste de contact, relayé par son ParentNode, afin de confirmer sa localisation et son état OnLine. Il peut aussi recevoir des demandes d'initialisation de session toujours provenant de « clients amis ». Il est donc préférable pour le client de passer en mode « threadé ».

- Récupération de la liste des contacts

  Afin d'obtenir la liste des contacts associée à un compte, le client doit d'adresser à un ES. Ici aussi le client contient dans son code des adresses IP d'ES.
  Exactement comme pour les LS le client va essayer des connexions TCP successives jusqu'à établissement d'une connexion TCP avec le ES. Le même échange de paquets de type « HTTPS Mini HandShake » que avec un LS est effectué. Le client doit maintenant s'authentifier auprès de l'ES, et encore une fois

cela est fait de la meme manière qu'avec un LS, aux différences près que moins de paremètres (version du client, langue du client, etc..) sont envoyés à l'ES, et que l'ES en réponse n'envoie qu'un code décrivant le succès ou l'échec de l'authentification.
Une fois authentifié auprès de l'ES, le client envoie a celui-ci une demande de liste de hash. L'ES va renvoyé en réponse, une liste d'objets contenants autant de hash CRC32 que d'éléments contenus dans la liste de contact.
En effet chaque hash est associé à un élément de la liste de contact. Et chaque hash est en fait un hash CRC32 du DirBlob contenant les informations détaillées de l'élément de la liste de contact.
Si le client ne reconnait pas un hash dans son cache de contact (cache évidemment vide lors de la première connexion du client) il va envoyer à l'ES une requête « GetHashDetails » avec comme paramètre le hash inconnu. L'ES va donc répondre au client en lui envoyant un paquet de type DirBlob contenant comme charge utile les informations détaillées (Pseudo, Nom Réel, Pays, Ville, Date de création du compte, Statut du lien avec l'utilisateur etc..) du contact associé au hash inconnu.
Ainsi le client récupère les détails de chaque élément de sa liste de contact.
Il peut alors se déconnecter de l'ES.
Notons qu'un élément de la liste de contacts peut etre soit un autre utilisateur Skype, soit un groupe de contacts, soit l'adresse email associé au compte. Un champ 'displayname' contenu dans le DirBlob permet de définir la nature de chaque élément en vérifiant le premier caractère de ce champ la. Par exemple pour un utilisateur le champ vaudra : « u/toto », pour

l'email associé au compte « p/email@toto.com », ou pour un groupe « g/B1F0CDE2 ».

- Localisation des utilisateurs de la liste de contacts

  Le client va maintenant afin de savoir l'état OnLine de chaque utilisateur de sa liste de contact, interroger le « Global Index[1] » qui est en fait une méthode pour retrouver le DirBlob contenant la localisation d'un client connecté ayant broadcasté sa présence récemment sur le réseau.
  Le client devra donc pour chaque utilisateur de la liste, calculer l'ID du slot associé au login du contact, ensuite demandé a son ParentNode de lui renvoyé au moins 5 adresses IP de SN appartenant à ce slot là. Une fois ces adresses connues il enverra à chacun des SN, via UDP, une requête de recherche de contact avec comme paramètre le login du contact. Si le contact en question a broadcasté sa présence dans son slot associé depuis au moins 72h ([1]), le client recevra en réponse des SN interrogés le DirBlob de localisation broadcasté par l'utilisateur. A ce stade pour chaque contact le client connait donc pour chaque utilisateur une liste de localisations qui ont été celles utilisées par l'utilisateur dans les 72 dernières heures. Mais toutes ne sont plus valides, car le client n'est plus forcément connecté depuis chacune de ses localisations.
  Il faudra donc lui envoyer un ping directement ou a son ParentNode suivant sa joignabilité. Le client envoie donc au relais ou directement à l'adresse publique une demande ping avec comme paramètre le login de la session. En cas de réponse du relais ou du client contacté directement, on sait alors que celui-ci est en ligne à la localisation pingée.

- Réponse aux pings pour valider la présence sur une localisation

  Comme préciser plus haut, en l'état REGISTER le client doit resté en écoute de son ParentNode ou en écoute directement sur un port publiquement accessible si possible afin, car il peut recevoir en provenance d'utilisateurs amis, des demande ping dans le but de confirmer sa présence sur une localisation.
  Le cas échéant le client doit , s'il veut etre visible OnLine par les utilisateurs amis, répondre en renvoyant à la provenance du ping (soit un relais, soit contact direct) un paquet de type « pong », avec comme paramètre son état (Disponible, Occupé, etc..) (Hypothèse a vérifier).

ANNEXE

ADRESSES DE BOOTSTRAP

```
Host  Hosts[] =  {{"193.88.6.19", 33033},
                  {"194.165.188.82", 33033},
                  {"66.235.180.9", 33033},
                  {"66.235.181.9", 33033},
                  {"212.72.49.143", 33033},
                  {"195.215.8.145", 33033},
                  {"64.246.48.23", 33033},
                  {"64.246.49.60", 33033},
                  {"64.246.49.61", 33033}} ;

Host  LoginServers[] = {{"194.165.188.79", 33033},
                        {"193.88.6.13", 33033},
                        {"212.72.49.141", 33033},
                        {"80.160.91.5", 33033},
                        {"195.215.8.141", 33033}} ;

Host  EventsServers[] = {{"212.72.49.142", 12350},
                         {"195.215.8.142", 12350}} ;
```

CLES PUBLIQUES STOCKEES DANS LE BINAIRE SKYPE :

```
char  *SkypeModulus1536[] =
"c095de9e868dc9fe1de94ab5e38e8dbdcae7e6249ef147de503f3e1c76e65af06f7714872f
3527ee16410170196e4b2277db206827505bc48b4b63159f8eb0d56d14de686e5f6840e3252
2f8b7dafc6b901b83495757f4269b59440bc7824d4543eae2b00b9d4d21b0b056ae53d6cc6c
3a35a5b10e72710cad00db5a42903e277e361cd1761a074afe997cc4a2c77427854ea1176da
481cadb981ee145711c7160c5b31f5194f64ec919bf57dc544656f39bad7bbdacb6e46f22c3
0173df2ea7",
"a8f223612f4f5fc81ef1ca5e310b0b21532a72df6c1af0fbec87304aec983aab5d74a14cc7
2e53ef7752a248c0e5abe09484b597692015e796350989c88b3cae140ca82ccd9914e540468
cf0edb35dcba4c352890e7a9eafac550b3978627651ad0a804f385ef5f4093ac6ee66b23e1f
8202c61c6c0375eeb713852397ced2e199492aa61a3eab163d4c2625c873e95cafd95b80dd2
d8732c8e25638a2007acfa6c8f1ff31cc2bc4ca8f4446f51da404335a48c955aaa3a4b57250
d7ba29700b",
"aa4bc22ba5b925573c48bf886efad103a37d697283a3d37b8bf5a3eb2d09a9ae73e6905fcb
3c6af06e4170b7a1856c70eab972cd3a468a28d7a87ceaad2404126dd5843f8895a0cfd9b07
085afe60b8ae391a703479846d800737ab02fca5ead5673f416d5fb4a95e1d27bb4b7ca5411
e74e98623f563b1d3709d749b3ee6d87242a8da04f39fd61ea679acc8e28601dadcc4434918
dc544cd365f7b897600b3fb62875f4517ae32601764ae8d28b924074e47ebc2bcaaa42d9d8f
eb82137787";

char  *SkypeModulus2048[] =
"b8506aeed8ed30fe1c0e6774874b59206a77329042a49be2403da47d50052441067f87bcd5
7e6579b83df0bade2beff5b5cd8d87e8b3edac5f57fabccd49695974e2b5e5f0287d6c19ecc
31b4504a9f8be25da78fa4ef345f91d339b73cc2d70b3904e11ca570ce9b5dc4b08b3c44b74
dc463587ea637ef4456e61462b72042fc2f4ad5510a9850c06dc9a7374412fcadda955bd980
0f9754cb3b8cc62d0e98d8282180971055b457c06f351e61164fc5a9de9d83d1d1378964001
380b5b99ee4c5c7d50ac2462a4b7ea34fd32d90bd8d4b46410263673f900d1c60470165df9f
3cb48016ab8ca45ce6875a71d977915ca8251b50258748dbc37fe332edc2855",
```

```
"bfe2db4bdf48358d3ae7c7d260989e4c7c1d81e6d9bd62e3a34dcaaa55f16c8c976a8862bc
2462ae1b4063fd3d0de97e8aee5b793e171e22136319ff3474667c47ed1c14a892a19644005
401cef46b5a8bd9510b3c8eeff80755decd2932cdca9119d96c5e16f6603b24ccc062822ca2
50336915a77aeab693268e1e22b1209d22f65a54cf7bcc7a4f9677eab54df8c33b16755ce1c
f907efae988df8a4b4fad0591d95617189cd5ae0ef0935b2b458d72c7cc3de19a6268af4b06
a8ce324292ca01b6a3adf7d133101a1e6659b03e0a1e14f07f8a11f5ec945c1a6f1eff06e75
764262749acd6c3bf59503d970b038f77018cb09eb924fc2045db9b3248d6d5",
"9f2a6038e8132c3545bc7c5fbe56903e08e83353b1defe7042b5c8457ba5a40254b2812a84
3be54993dafed5fa55a415be652e2ff533483a1ab3907c5603df47a59311973c2b4c39cf942
a58f12149ce0437b341aa58add6dc2ab27800f1f529975506c10925e5254ff0766ed276b194
d36c783a7d426672e32e7d6881c91e167ccbb38f4f9ee703d621712b7de384c6c97fe5cf557
c839d47ebf1c45db2ec554b4aebf203d00896e9b202e564251f6fc623f6810208ffd1497c4d
e673d87d693febc17ec5157aa5aa8a0055976fd4817b6ccef76d4525741820d72ef89c25589
71189c5e42cdeb271590f6bb0e2bede43a28003f54298be689a39791ec8a919",
"bf137933ab269e3bacbea76da16f6ff06748028e79efe20aa696368fc88f6abca4571790f3
d67ea4b6f382850f8f27a2ee4c044be541083d74ecbb30142070a7bc2746ff01fb243c4ec2a
5fba296c3fb9c357429b4d511741bd2bf97034d29fe4fbd9013df87b725941e9696635fd53c
af2f56b7b3609c51b8448256c4fb6078dde89f06d54c8bb4e2f6fbfb6ee42e698649e32e74d
c542c94c2be3f9833ef22f988f9aa9c4fa5267b8aaf455ef06c693a36ca4447763e86508607
0713328e2f835649e1a15d78eaa6f90f9b5a6fac3f2eb3e24979bdacc4843956ef44c186f50
07278c7d6cc1ceaefd8c56121ffa39c8d75873ae5a853b7cd1002885b49b209",
"d7edafe450023e3a4b0f7f1925196943ce1defd4d158cc868a29973bbedb9b90d203331b01
c9752f254d7c922f8b5bdba2d25bd5955b887f26532afda73e37527f1265925ddbb6f1ed5ef
d660b1cad6ecb94628e1c90311a320db1a758ec36e3df926df48ebc1c099714dfe5e7541eec
3ce96dd7d6f8b55288c081646c51a6b77f5c1ef1025b7862ee9d5e7d491293471ae7159daa4
0ccd7db2753071c11264aa5cc861d5fedc8fed6f2503b0a68a17d78cdfe3f698bd8af21ac11
52f2dcef7c515bd9c9e4934cd9c9977ade4c77fbcd184111332f2737847b5ee713b6568d55c
14618ae1e4ce088431709d9ad8fae987887ac829c3f7968bee8d1cfd347e341",
"d52af9c4194936e76ca66fd276533926d073b261c8de4cba4ea8f758dcce63e1c391220085
ac94492926fde61ac0ca4b7b1acdd90a4d601b242cd2f890d8f826ab0c269bf1d23f497f2c2
36f1f57c01b881b88616eab2aa2028d533bb449b579063fcd287fba64ab0d64f4c1f38845d2
2017be45b7942348b494b1dd73982719be5ee37384b9feb41473e8bf71c8c4edab7a4869078
79e6a797baa2d7d9462fc24a7ad6a00598646064beb19f27513bee559f0316a398eaa5d5e61
8a41fc0d16ee1c6cf17e84bf7571227fe98a29d205c0ffd5546bed9f87e22e49ebee6d58e90
8e268d258d7520589f2b730d72639bb751ac4fb4d39125bb5b205bacfb423df",
"a4deafb40a982a87c989fe919b888dfe185113b17106b3241e7d166177b322b790131ddeba
e8d878bd8ae7e2e39794cab70988778cb13d121528e260eedabb3df35a320a02b7e68b63d4c
d143d0ef93ab565d2163e467b77ae4644e9850e922fcdc61c58020e236a70557188e8ff2c33
2ff9e4f3d984042a2c802917ba9bcdf9277b4de3bf7055423bcac61332f2983d3fd9074dcb6
10e4c8d751bb38d4c20b470a8314d44790462e62c3b45a77cc290aab52b0a237fccd7513b5e
8fd50ec4e79dac5047191417ffd68ebfa023f275a0fff93531cdda7b6287d34e41e89227c29
42b6aeca11956fdb566f65be7f1c456b1b07b731cb78698b08478f5e7c856cb",
"a903c6079aac211634af60c85955889da69a552a2e994ef9395a53fc258c78c6308961d7bf
3e87dd85ebb55087ffa151433514901e5b8e79458209864e82d839de8793002e15528d979a4
b853a47a7d74e463ebdc321827308559c68a81f6de72660625f577ac42fc0addd31ad869715
5de21d977bd9874ce65a6232cad0c08a21b9a2a3d4d14923115ff0d9b5f09e645f4d3a6c45c
a1838a8f7a519b2d82fd82282678260d934bc8d42314ac6c703189ad734b2e9c8d285a50447
fda2d01bd26452e63ce290ca1f88b1237abdd942372a384902ab3495a37bfd68c45d46008d1
34984f06f9114d411bd560fe48ec3a16a3f4ff0127ca52928f3261ddbb76b9d",
"b4792d7289d6aaba727b338d6ed91f7fa34f9d9abca23795bff72f7f38f4c27b4161494898
91c2b615d0b5db19a09d6462d26ee12526f93f4bd129655ecc890727f8a6ec71c17a82b02f6
8afe8dab6e40620236753784d4a279159dc613b2eebc1bbf9fa65b8ca1d13ded28ea575c591
738772369ca4049329d10c06c194e92e13394f8b53c890235b922f18b24ccc71d59065aa6e4
5889da11d33feee17bd994c456d7635238f431403badb45b9ea907e4385ae2ef178dd5154c7
7895e9a8f60eef3c3969b356a0c11e6d3d4d0563c1e76d8268117f0eb4c160b3b941e77518a
6361d17241f28c7f84ee0e378a831782e9dc688584a0ef9b383835e202bd67f";
```

```
char  *SkypeModulus4096[] =
"c3ac2b9912720fd9d5d121570bb8acb6c5b9b5ad4af08b75a68728860759a256f1c4357a32
9ac61528e968b7b41cae0f7c61784e0e72e465828fbb4e92931a2e14856d4a9600045df661c
ca37ccd6b993c93fcf62c0eab46882460df70b6f866293bd6d8fe35bcbd9c63ca4909c30652
933d2b1943e67633af7ad0599ac8a61b914e2b97d9b1736683f1457695842951fff68e36fc1
70a54ddbb6b4f7d0361f5a6c36ad2837281aa9359460e6e15aa84d0e17ba9f1a9de3c88cc92
33e371a6f18e48892e570679be601ddc134669967e2c550895ed97e49189b7164997eb26654
0b6b19d259188ddb9224ee846d1d2ca4e86eba4be39a7b1662f7e230b9d8a8f2553f3394a8b
6f0668c3372bbba713064543b8f932ca34b3eb44bd8533191550b06960553a0e69386770569
c1bc087d21976ef570f6634a5dd127407af32f39a56f5e72aed3590fa4722dc0a55e475b2d1
ab39b2f0904168f97a16f4762d6804f7164feecc1a57823a3786c53bcdd73a7086ecd56142a
f052b86dec233ef8abeab437436c6c1c6ce231aa93bdf669f25c1c30b12df932df4b9662f90
b1e1aaeca77a6c773dcb28669cf1852b8c14615e52905ccf3beac1546f499778dd01fc06087
6afe3f5f97de6d3cccae7a686222c17699f5968c453fd6f692e8254961018f5986d98c01c35
7d624d3d6c44e3b4ca344de00b2d8bf3939c2758b057bd4985",
"ba7463f3b6ccfd1327750915654e4823a5594b656cb9114b17146425bf5d720c9d94ca0bd2
ca74ee8ac1af7940950ceebd161b1a631e4a6c98015a056d7af2decc2fc582a0b04ccb5eebb
06a438816fbc594de942f9a90fb5c89bdcf215f0ddd59afba36b03538f5ef348dd963669004
19c488263dbc547f26ffb54e9fb8faa5a9fed038bc2bd3b14a6b4ffaa5e01e84a029df338e4
7b267e15976508abb83b4bf88c523b5af7ba597fc4f4b07da84cd4eafb06cd9dee246854bb8
2be2620981e0c1491d0ed9ad67810cbf1ee7ad7d89a4c66ecb24f10d1c57e7a6947566a8a97
39b2ca0af160761cddb4bf279ff6668a46c6c38a365dfeac5ba0f130418fe6234ae0642bff1
8c9a3422b0ef0f64373f7414ccf80cfd5440b3752de911542cdc10a25899ef7ff7ef32e8d66
26caf5d8887d593a7323d663d5c1cd4885b48f3adfa529c42c9177519a2c283db8b68791dd8
020cf2fb047fcf58d4605d3891074598eadf1b9c8da1a88b7fd7635d419c16ad109ed288f46
1d64325ae4929298c204e877515417645057bca10c6ceb3de4bd63441f2e2e2b03bad33781b
f4e242f30da3e2407736668ced9193e022b75ab4f65d7201999fa317e9ad30a80080797279b
febd89614827c323979ec5141a25c15bf800f337907fe50b0cdb9eb07b9665958cc9373188d
d58eb6571f1f41c4aa7b6357138557a37d122ab6c3f7cd2de9";
```