Cognome Nome Matricola

Rispoli Mario 0622701195

Ruggiero Valeria 0622701250

Russo Gabriele Luigi 0622701175

Sabbarese Francesco 0622701190

Silla Marta 0622701337

# PROPOSTA DI PROGETTO (COSÌ COME INDICATA A INIZIO MAGGIO)

============================================================

Titolo Progetto: Grafo di prossimità

============================================================

Tema del progetto:

Utilizzo del protocollo DP3T e dei dati da esso trattati al fine di realizzare un grafo di prossimità dei pazienti risultati infetti e degli utenti con i quali sono entrati in contatto.

============================================================

Entità coinvolte:

Governo: gestisce un server che permette di coordinare le informazioni collezionate dagli smartphone. Coordina anche i laboratori per assistere i pazienti nelle loro operazioni di comunicazione al sistema della loro positività.

Laboratorio analisi: il posto in cui può recarsi un cittadino per effettuare dei test e verificare la positività al virus.

Studiosi di epidemiologia: scienziati coinvolti nell'analisi dei dati rappresentati dal grafo di prossimità ai fini di studiare la diffusione del SARS-Cov-2.

Cittadino infetto: un cittadino risultato positivo secondo la certificazione di un Laboratorio di analisi.

Cittadino a rischio: un cittadino che risulta aver avuto un contatto con un cittadino infetto tale da essere considerato a rischio di contagio.

============================================================

Dizionario:

Grafo di prossimità: un grafo che descrive i contatti avvenuti tra i pazienti infetti e quelli che (su base volontaria) hanno comunicato di essere entrato in contatto con loro. Ciascun ramo rappresenta un contatto avvenuto ed è adiacente ad almeno un paziente infetto. Il ramo è caratterizzato da informazioni utili.

Falso contatto: un contatto mai realmente avvenuto.

============================================================

Obiettivi del progetto:

Realizzazione di un protocollo che permetta di realizzare un grafo di prossimità dei pazienti risultati infetti e degli utenti con i quali questi sono entrati in contatto. Il protocollo è costruito sulla base di quanto descritto nel protocollo DP3T. Lo scopo del progetto è quello di fornire uno strumento utile a studi epidemiologici. Ciascun utente è libero di decidere se contribuire, attraverso i suoi dati, alla realizzazione di questo strumento. I cittadini che decidono di contribuire non devono essere identificabili, si vogliono infatti mantenere le caratteristiche di tutela della privacy garantite da DP3T. Si vogliono scoraggiare possibili tentativi di generazione di falsi contatti.

============================================================

Task previsti:

1) Descrizione dettagliata (ma non completamente formale) delle proprietà di completezza, sicurezza e privacy desiderate.

2) Progettazione di un protocollo che permetta la costruzione corretta di un grafo di prossimità.

3) Analisi euristica ma sufficientemente ricca di dettagli che validi la bontà della progettazione rispetto alle proprietà elencate nel punto 1).

4) Implementazione parziale del protocollo.

============================================================

Note aggiuntive:

Il white paper di DP3T parla del grafo di prossimità, tuttavia non dice come le informazioni vadano gestite, nè dice se sia o meno previsto un protocollo per validare la richiesta di un utente di essere aggiunto al grafo dopo aver dichiarato un contatto con un infetto.

Criticità:

Sia A un utente che ha comunicato al server, secondo protocollo, di essere infetto. Sia B un utente che ha avuto un contatto con A e sia C un utente che non ha avuto un contatto con A. B e C possono entrambi comunicare al server di essere entrati in contatto con A e non c'è modo che il server, che in questo momento conosce solo gli ephID e la chiave di A, possa stabilire la veridicità dell'evento.

Tra le possibili idee che potremo seguire, forniamo la seguente BOZZA:

--------------------------------------------------------------------------------------------------

Affinché il grafo sia una rappresentazione veritiera dei contatti realmente avvenuti, si è pensato in particolare, all'introduzione di un codice che attesti un'avvenuta connessione tra due utenti e al contempo non minacci la loro privacy.

Quando B vuole comunicare di essere stato in contatto con A, deve caricare sul server anche il codice del contatto associato all'ephID di A comunicatogli dal server.

Il protocollo di validazione del codice inoltre non deve fornire al server informazioni ulteriori a quelle in suo possesso, se non strettamente necessarie. Il protocollo dunque potrebbe prevedere che il server chieda ad A di verificare il codice comunicatogli da B. Si vuole infatti evitare che il server possa dedurre indesideratamente contatti, se non esplicitamente su richiesta degli utenti interessati, come nello scenario appena descritto. Si vuole infine far notare che il codice cambia da contatto a contatto, anche se tra i medesimi utenti, per le stesse ragioni mosse nel contesto degli ephID. B continua a non avere ulteriori informazioni riguardo A. A continua a non avere ulteriori informazioni riguardo B. Nel contesto in cui ci fossero avversari che potrebbero mettersi in ascolto delle connessioni, al fine di preservare la segretezza del codice scambiato tra A e gli altri utenti, si potrebbe procedere a una cifratura del codice.

--------------------------------------------------------------------------------------------------

Si prevede, al di là dei dettagli del protocollo suggerito nella bozza, un meccanismo di autenticazione del contatto tra A e B che permetta al server di stabilire se B è realmente entrato in contatto con A, e di conseguenza decidere se aggiungere o meno B al grafo di prossimità, cercando di sostenere il livello di privacy fornito da DP3T.

# DISCUSSIONE SULLE PROPRIETÀ DI CONFIDENZIALITÀ E INTEGRITÀ CHE SI INTENDONO GESTIRE E DESCRIZIONE DEGLI AVVERSARI

## OBIETTIVO

Realizzazione di un protocollo di appoggio a DP-3T che permetta di realizzare un grafo di prossimità dei pazienti risultati infetti e degli utenti con i quali questi sono entrati in contatto. Il fine è fornire uno strumento utile a studi di natura epidemiologica. In particolare, quando un’autorità sanitaria analizza un test positivo, il cittadino infetto, come utente, ha la possibilità di comunicare al server il proprio stato di salute. Il server provvederà a notificare tutti gli altri utenti, che saranno in grado di dedurre se sono o meno entrati in contatto con l’utente malato. Essi potranno quindi decidere di collaborare col server per tracciare un grafo di contatti, noto anche come grafo di prossimità.

### ENTITÀ COINVOLTE

|  |  |
| --- | --- |
| *Server*: | si occupa di analizzare i messaggi ricevuti da utenti malati e utenti a rischio affinché sia possibile mapparli sul grafo di prossimità in seguito a verificato incontro. Inoltre, notifica agli utenti le informazioni strettamente necessarie affinché siano capaci di dedurre un possibile contatto con un utente infetto; |
| *Utente malato*: | utente in possesso di un codice di autorizzazione rilasciatogli da un’autorità sanitaria fidata in seguito alla positività al test *COVID*; |
| *Utente a rischio*: | utente che risulta aver avuto un contatto comprovato con un utente malato; |
| *Avversario*: | chiunque, indipendentemente dallo scopo, possa sfruttare le vulnerabilità del sistema per compromettere la confidenzialità e l’integrità dei dati da esso gestiti; |
| *Autorità sanitaria*: | istituzione che rilascia agli utenti malati un codice di autorizzazione che attesta la positività al test *COVID*. |
| *Istituzione giuridica*: | istituzione che ha il potere legale di richiedere rispettivamente al server l’accesso al grafo di prossimità e all’autorità sanitaria le corrispondenze tra identità dei pazienti e codici di autorizzazione a loro rilasciati. |

### CONFIDENZIALITÀ

L’identità degli utenti che decidono di contribuire alla costruzione del grafo non deve essere esposta. In particolare:

* l’identità di un utente malato deve rimanere segreta all’applicazione;
* le identità degli altri utenti con cui un malato entra in contatto devono rimanere segrete all’applicazione;
* gli archi tra i nodi mappati sul grafo di prossimità non devono poter essere utilizzati come prova per attestare un reale incontro tra due o più utenti anche qualora la loro identità fosse conosciuta da terzi.

### INTEGRITÀ

L’identità degli utenti che decidono di contribuire non dev’essere esposta. Si vogliono infatti mantenere le caratteristiche di tutela della privacy garantite da DP-3T.

Gli archi presenti sul grafo di prossimità devono mappare incontri realmente avvenuti tra due o più utenti.

## 3. PROGETTAZIONE DEL SISTEMA

### INTRODUZIONE

Il sistema proposto estende la seconda versione del protocollo DP-3T descritta in questo [white paper](https://github.com/DP-3T/documents/blob/a0a88c3efebb40640a455886f7c63d7cc748f7e9/DP3T%20White%20Paper.pdf) e denominata *unlinkable* per sottolineare il maggior livello di privacy che offre ai suoi utenti rispetto alla sua prima versione.

### *UNLINKABLE* DESIGN DP-3T

Il design prevede che diversi ephIDi siano generati randomicamente e indipendentemente a partire da un seedi come segue:

ephIDi = TRUNCATE128(H(seedi))

dove H è una cryptographic hash function (i.e. Sha-256).

Gli smartphone salvano in locale i seedi usati in una finestra temporale significativa per mappare i contagi (i.e. 14 giorni).

In particolare, seedi è una stringa random di 32 byte di durata L, intervallo di tempo dopo cui viene generato seedi+1.

Quando un utente viene testato risultando positivo, può scegliere quali ephIDi comunicare al server: può per esempio decidere di non comunicare gli ephIDi che la sua applicazione ha generato in un determinato intervallo di tempo.

Un utente infetto può procedere quindi con l’upload sul server delle stringhe seedi||i da lui filtrate e selezionate.

Il server dunque computa H(TRUNCATE128(H(seedi)) ||i) = H(ephIDi||i) e manda il risultato, inserito in un Cuckoo filter, in broadcast.

In questo scenario, lo smartphone controlla se ci sono dei match tra le H(ephIDi||i) che sono presenti nel filtro e le H(ephIDk||k) che ha salvato localmente.

La versione del 12/04/2020 del white paper di DP-3T contiene accenni circa la costruzione di un grafo di prossimità. Il server potrebbe costruire tale grafo mappando il contatto tra un utente infetto e un utente che ha riscontrato un match con l’H(ephIDi||i) che il server stesso ha inviato in broadcast.

Tuttavia, nella versione del 25/05/2020, gli autori del documento dichiarano di non volersi più curare della condivisione dati a scopo di ricerca.

Risulta comunque evidente la necessità di instaurare un protocollo che abbia il fine di garantire l’integrità di costruzione del grafo validando o meno le richieste mosse dagli utenti.

### FEATURES AGGIUNTE

Il sistema proposto aggiunge al protocollo di base un meccanismo che consenta sia agli utenti di comunicare al server un avvenuto contatto con un utente infetto sia al server di gestire la costruzione e l’evoluzione del grafo di prossimità riconoscendo un falso contatto da un autentico.

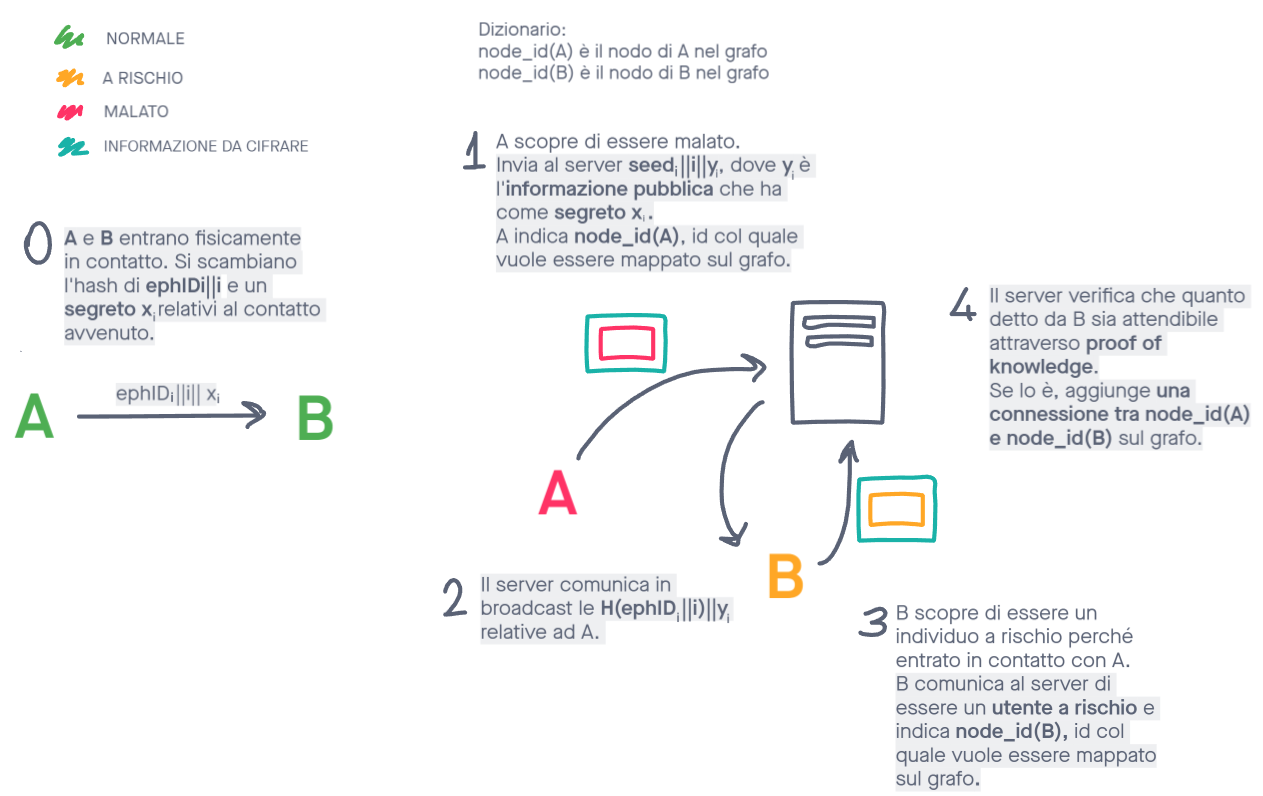
In particolare, viene utilizzato il protocollo *Schnorr non-interactive Zero-Knowledge* *proof (G,q,g,x,y)* per dare prova al server di un contatto realmente avvenuto.

Dal momento in cui l’applicazione viene installata sullo smartphone di A, DP-3T prevede la diffusione in broadcast di messaggi BLE contenenti le stringhe ephIDi||i||xi.

Successivamente, A, verificatosi malato, fa l’upload sul server di seedi||i, come già descritto da DP-3T in *unlinkable*, e concatena l’informazione pubblica yi come segue: seedi||i||yi.

Quando l’utente B vorrà dar prova al server dell’autenticità di un suo contatto con A, stabilirà con esso un protocollo di *zero knowledge proof* che vedrà come segreto xi e come informazione pubblica yi.

Il server è quindi capace di stabilire se mappare o meno B sul grafo di prossimità in base all’esito della *zero knowledge proof* conclusasi al passo precedente.



1. B riceve ephIDi||i||xi mandato in broadcast da A dove:

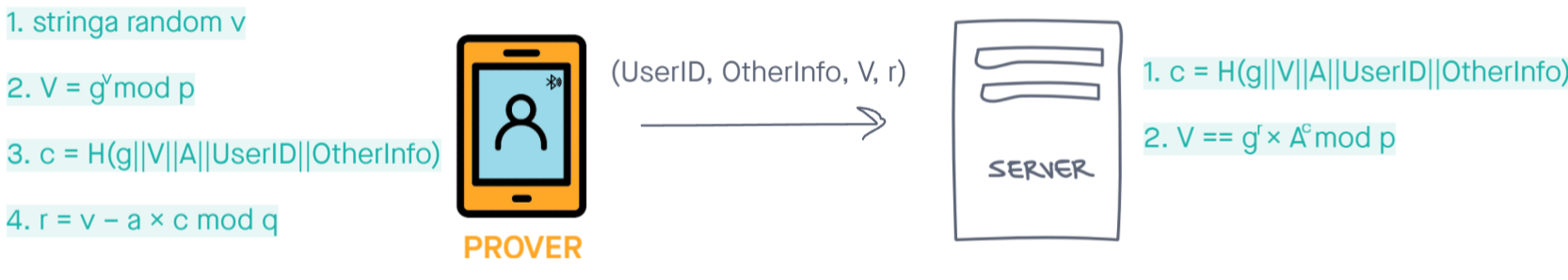
* ephIDi = TRUNCATE88(H(seedi))
* |xi| = 40 bit
* |(ephIDi||i)| + |xi| = 128 bit = 16 byte

1. A scopre di essere malato e invia al server seedi||i ||yi.
2. Il server invia in broadcast il Cuckoo filter contenente le H(EphIDi||i)||yi che ha ricevuto da A.
3. B scopre di essere stato in contatto con A trovando un match con H(ephIDi||i). Può dare prova al server del reale contatto, attraverso il protocollo di *proof of knowledge* risolvendo yi con la chiave segreta xi.
4. Il protocollo di *proof of knowledge* stabilisce l’autenticità del contatto tra A e B permettendo al server di aggiornare il grafo di prossimità.

### ZERO KNOWLEDGE PROOF (G,p,g,q,x,y)

In accordo al modello *“Schnorr non-interactive Zero-Knowledge* *proof”* il protocollo di verifica di un contatto prevede che un utente mandi al server su connessione https (UserID, OtherInfo, a, z) dove:

* g: è il generatore del gruppo G di ordine p
* p: è un numero primo molto grande
* q: è un grande divisore primo di p-1
* a = gr mod p, con r una stringa random in [0, q-1]
* c = H(g||a||yi||UserID||OtherInfo)
* z = (r – xc) mod q
* x: è il segreto che nel documento compare come xi
* yi = gXi
* UserID: è l’identificativo univoco del prover, che nel documento compare come node\_id;
* OtherInfo: è il certificato del server
* H: è una cryptographic hash function (i.e. SHA-256) che ha come dimensione dell’output il minimo ordine di q



Come pubblicato [nel rfc5114 *“Additional Diffie-Hellman Groups for Use with IETF Standards*” da M.Lepinski S.Kent BBN Tecnologies”](https://www.ietf.org/rfc/rfc5114), ci sono tre gruppi standard proposti dal *National Institute of Standards and Technology (NIST)*:

* 1024-bit MODP Group with 160-bit Prime Order Subgroup
* 2048-bit MODP Group with 224-bit Prime Order Subgroup
* 2048-bit MODP Group with 256-bit Prime Order Subgroup

#### 1024-bit MODP Group with 160-bit Prime Order Subgroup

The hexadecimal value of the prime is:

p = B10B8F96 A080E01D DE92DE5E AE5D54EC 52C99FBC FB06A3C6

9A6A9DCA 52D23B61 6073E286 75A23D18 9838EF1E 2EE652C0

13ECB4AE A9061123 24975C3C D49B83BF ACCBDD7D 90C4BD70

98488E9C 219A7372 4EFFD6FA E5644738 FAA31A4F F55BCCC0

A151AF5F 0DC8B4BD 45BF37DF 365C1A65 E68CFDA7 6D4DA708

DF1FB2BC 2E4A4371

The hexadecimal value of the generator is:

g = A4D1CBD5 C3FD3412 6765A442 EFB99905 F8104DD2 58AC507F

D6406CFF 14266D31 266FEA1E 5C41564B 777E690F 5504F213

160217B4 B01B886A 5E91547F 9E2749F4 D7FBD7D3 B9A92EE1

909D0D22 63F80A76 A6A24C08 7A091F53 1DBF0A01 69B6A28A

D662A4D1 8E73AFA3 2D779D59 18D08BC8 858F4DCE F97C2A24

855E6EEB 22B3B2E5

The generator generates a prime-order subgroup of size:

q = F518AA87 81A8DF27 8ABA4E7D 64B7CB9D 49462353

Test values:

xA = B9A3B3AE 8FEFC1A2 93049650 7086F845 5D48943E

yA = 2A853B3D 92197501 B9015B2D EB3ED84F 5E021DCC 3E52F109

D3273D2B 7521281C BABE0E76 FF5727FA 8ACCE269 56BA9A1F

CA26F202 28D8693F EB10841D 84A73600 54ECE5A7 F5B7A61A

D3DFB3C6 0D2E4310 6D8727DA 37DF9CCE 95B47875 5D06BCEA

8F9D4596 5F75A5F3 D1DF3701 165FC9E5 0C4279CE B07F9895

40AE96D5 D88ED776

xB = 9392C9F9 EB6A7A6A 9022F7D8 3E7223C6 835BBDDA

yB = 717A6CB0 53371FF4 A3B93294 1C1E5663 F861A1D6 AD34AE66

576DFB98 F6C6CBF9 DDD5A56C 7833F6BC FDFF0955 82AD868E

440E8D09 FD769E3C ECCDC3D3 B1E4CFA0 57776CAA F9739B6A

9FEE8E74 11F8D6DA C09D6A4E DB46CC2B 5D520309 0EAE6126

311E53FD 2C14B574 E6A3109A 3DA1BE41 BDCEAA18 6F5CE067

16A2B6A0 7B3C33FE

#### 2048-bit MODP Group with 224-bit Prime Order Subgroup

The hexadecimal value of the prime is:

p = AD107E1E 9123A9D0 D660FAA7 9559C51F A20D64E5 683B9FD1

B54B1597 B61D0A75 E6FA141D F95A56DB AF9A3C40 7BA1DF15

EB3D688A 309C180E 1DE6B85A 1274A0A6 6D3F8152 AD6AC212

9037C9ED EFDA4DF8 D91E8FEF 55B7394B 7AD5B7D0 B6C12207

C9F98D11 ED34DBF6 C6BA0B2C 8BBC27BE 6A00E0A0 B9C49708

B3BF8A31 70918836 81286130 BC8985DB 1602E714 415D9330

278273C7 DE31EFDC 7310F712 1FD5A074 15987D9A DC0A486D

CDF93ACC 44328387 315D75E1 98C641A4 80CD86A1 B9E587E8

BE60E69C C928B2B9 C52172E4 13042E9B 23F10B0E 16E79763

C9B53DCF 4BA80A29 E3FB73C1 6B8E75B9 7EF363E2 FFA31F71

CF9DE538 4E71B81C 0AC4DFFE 0C10E64F

The hexadecimal value of the generator is:

g = AC4032EF 4F2D9AE3 9DF30B5C 8FFDAC50 6CDEBE7B 89998CAF

74866A08 CFE4FFE3 A6824A4E 10B9A6F0 DD921F01 A70C4AFA

AB739D77 00C29F52 C57DB17C 620A8652 BE5E9001 A8D66AD7

C1766910 1999024A F4D02727 5AC1348B B8A762D0 521BC98A

E2471504 22EA1ED4 09939D54 DA7460CD B5F6C6B2 50717CBE

F180EB34 118E98D1 19529A45 D6F83456 6E3025E3 16A330EF

BB77A86F 0C1AB15B 051AE3D4 28C8F8AC B70A8137 150B8EEB

10E183ED D19963DD D9E263E4 770589EF 6AA21E7F 5F2FF381

B539CCE3 409D13CD 566AFBB4 8D6C0191 81E1BCFE 94B30269

EDFE72FE 9B6AA4BD 7B5A0F1C 71CFFF4C 19C418E1 F6EC0179

81BC087F 2A7065B3 84B890D3 191F2BFA

The generator generates a prime-order subgroup of size:

q = 801C0D34 C58D93FE 99717710 1F80535A 4738CEBC BF389A99

B36371EB

Test values:

xA = 22E62601 DBFFD067 08A680F7 47F361F7 6D8F4F72 1A0548E4

83294B0C

yA = 1B3A6345 1BD886E6 99E67B49 4E288BD7 F8E0D370 BADDA7A0

EFD2FDE7 D8F66145 CC9F2804 19975EB8 08877C8A 4C0C8E0B

D48D4A54 01EB1E87 76BFEEE1 34C03831 AC273CD9 D635AB0C

E006A42A 887E3F52 FB8766B6 50F38078 BC8EE858 0CEFE243

968CFC4F 8DC3DB08 4554171D 41BF2E86 1B7BB4D6 9DD0E01E

A387CBAA 5CA672AF CBE8BDB9 D62D4CE1 5F17DD36 F91ED1EE

DD65CA4A 06455CB9 4CD40A52 EC360E84 B3C926E2 2C4380A3

BF309D56 849768B7 F52CFDF6 55FD053A 7EF70697 9E7E5806

B17DFAE5 3AD2A5BC 568EBB52 9A7A61D6 8D256F8F C97C074A

861D827E 2EBC8C61 34553115 B70E7103 920AA16D 85E52BCB

AB8D786A 68178FA8 FF7C2F5C 71648D6F

xB = 4FF3BC96 C7FC6A6D 71D3B363 800A7CDF EF6FC41B 4417EA15

353B7590

yB = 4DCEE992 A9762A13 F2F83844 AD3D77EE 0E31C971 8B3DB6C2

035D3961 182C3E0B A247EC41 82D760CD 48D99599 970622A1

881BBA2D C822939C 78C3912C 6661FA54 38B20766 222B75E2

4C2E3AD0 C7287236 129525EE 15B5DD79 98AA04C4 A9696CAC

D7172083 A97A8166 4EAD2C47 9E444E4C 0654CC19 E28D7703

CEE8DACD 6126F5D6 65EC52C6 7255DB92 014B037E B621A2AC

8E365DE0 71FFC140 0ACF077A 12913DD8 DE894734 37AB7BA3

46743C1B 215DD9C1 2164A7E4 053118D1 99BEC8EF 6FC56117

0C84C87D 10EE9A67 4A1FA8FF E13BDFBA 1D44DE48 946D68DC

0CDD7776 35A7AB5B FB1E4BB7 B856F968 27734C18 4138E915

D9C3002E BCE53120 546A7E20 02142B6C

#### 2048-bit MODP Group with 256-bit Prime Order Subgroup

The hexadecimal value of the prime is:

p = 87A8E61D B4B6663C FFBBD19C 65195999 8CEEF608 660DD0F2

5D2CEED4 435E3B00 E00DF8F1 D61957D4 FAF7DF45 61B2AA30

16C3D911 34096FAA 3BF4296D 830E9A7C 209E0C64 97517ABD

5A8A9D30 6BCF67ED 91F9E672 5B4758C0 22E0B1EF 4275BF7B

6C5BFC11 D45F9088 B941F54E B1E59BB8 BC39A0BF 12307F5C

4FDB70C5 81B23F76 B63ACAE1 CAA6B790 2D525267 35488A0E

F13C6D9A 51BFA4AB 3AD83477 96524D8E F6A167B5 A41825D9

67E144E5 14056425 1CCACB83 E6B486F6 B3CA3F79 71506026

C0B857F6 89962856 DED4010A BD0BE621 C3A3960A 54E710C3

75F26375 D7014103 A4B54330 C198AF12 6116D227 6E11715F

693877FA D7EF09CA DB094AE9 1E1A1597

The hexadecimal value of the generator is:

g = 3FB32C9B 73134D0B 2E775066 60EDBD48 4CA7B18F 21EF2054

07F4793A 1A0BA125 10DBC150 77BE463F FF4FED4A AC0BB555

BE3A6C1B 0C6B47B1 BC3773BF 7E8C6F62 901228F8 C28CBB18

A55AE313 41000A65 0196F931 C77A57F2 DDF463E5 E9EC144B

777DE62A AAB8A862 8AC376D2 82D6ED38 64E67982 428EBC83

1D14348F 6F2F9193 B5045AF2 767164E1 DFC967C1 FB3F2E55

A4BD1BFF E83B9C80 D052B985 D182EA0A DB2A3B73 13D3FE14

C8484B1E 052588B9 B7D2BBD2 DF016199 ECD06E15 57CD0915

B3353BBB 64E0EC37 7FD02837 0DF92B52 C7891428 CDC67EB6

184B523D 1DB246C3 2F630784 90F00EF8 D647D148 D4795451

5E2327CF EF98C582 664B4C0F 6CC41659

The generator generates a prime-order subgroup of size:

q = 8CF83642 A709A097 B4479976 40129DA2 99B1A47D 1EB3750B

A308B0FE 64F5FBD3

Test values:

xA = 0881382C DB87660C 6DC13E61 4938D5B9 C8B2F248 581CC5E3

1B354543 97FCE50E

yA = 2E9380C8 323AF975 45BC4941 DEB0EC37 42C62FE0 ECE824A6

ABDBE66C 59BEE024 2911BFB9 67235CEB A35AE13E 4EC752BE

630B92DC 4BDE2847 A9C62CB8 15274542 1FB7EB60 A63C0FE9

159FCCE7 26CE7CD8 523D7450 667EF840 E4919121 EB5F01C8

C9B0D3D6 48A93BFB 75689E82 44AC134A F544711C E79A02DC

C3422668 4780DDDC B4985941 06C37F5B C7985648 7AF5AB02

2A2E5E42 F09897C1 A85A11EA 0212AF04 D9B4CEBC 937C3C1A

3E15A8A0 342E3376 15C84E7F E3B8B9B8 7FB1E73A 15AF12A3

0D746E06 DFC34F29 0D797CE5 1AA13AA7 85BF6658 AFF5E4B0

93003CBE AF665B3C 2E113A3A 4E905269 341DC071 1426685F

4EF37E86 8A8126FF 3F2279B5 7CA67E29

xB = 7D62A7E3 EF36DE61 7B13D1AF B82C780D 83A23BD4 EE670564

5121F371 F546A53D

yB = 575F0351 BD2B1B81 7448BDF8 7A6C362C 1E289D39 03A30B98

32C5741F A250363E 7ACBC7F7 7F3DACBC 1F131ADD 8E03367E

FF8FBBB3 E1C57844 24809B25 AFE4D226 2A1A6FD2 FAB64105

CA30A674 E07F7809 85208863 2FC04923 3791AD4E DD083A97

8B883EE6 18BC5E0D D047415F 2D95E683 CF14826B 5FBE10D3

CE41C6C1 20C78AB2 0008C698 BF7F0BCA B9D7F407 BED0F43A

FB2970F5 7F8D1204 3963E66D DD320D59 9AD9936C 8F44137C

08B180EC 5E985CEB E186F3D5 49677E80 607331EE 17AF3380

A725B078 2317D7DD 43F59D7A F9568A9B B63A84D3 65F92244

ED120988 219302F4 2924C7CA 90B89D24 F71B0AB6 97823D7D

EB1AFF5B 0E8E4A45 D49F7F53 757E1913

### PROTOCOLLO DI COMUNICAZIONE

In relazione al design proposto bisogna definire due tipologie di pacchetti che vengono scambiati con protocollo HTTPS rispettivamente tra A e il server e B e il server.

|  |
| --- |
|  |
| *A -> Server* è il pacchetto inviato da A al server con cui A comunica di essere stato certificato infetto da un’autorità sanitaria:   * node\_ID: stringa random di 32 byte generata al momento dell’istallazione dell’applicazione con cui A viene identificato e mappato sul grafo di prossimità; * seedi||i||yi: concatenazione del seedi, della i e della yi; * authority\_authentication\_code: codice certificante la positività di A al test COVID. |

|  |
| --- |
|  |
| *B -> Server* è il pacchetto inviato da B al server con cui B dà prova al server di essere stato in contatto con A:   * node\_ID: stringa random di 32 byte generata al momento dell’istallazione dell’applicazione con cui B viene identificato e mappato sul grafo di prossimità; * contacted\_ephIDs: lista di H(ephIDi||i) associati ai contatti che B vuole comprovare. * prova contatto autentico: insieme dei messaggi che B, per ogni contatto, scambia con il server secondo il protocollo di *zero knowledge proof.* |

Use case: costruzione del grafo di prossimità

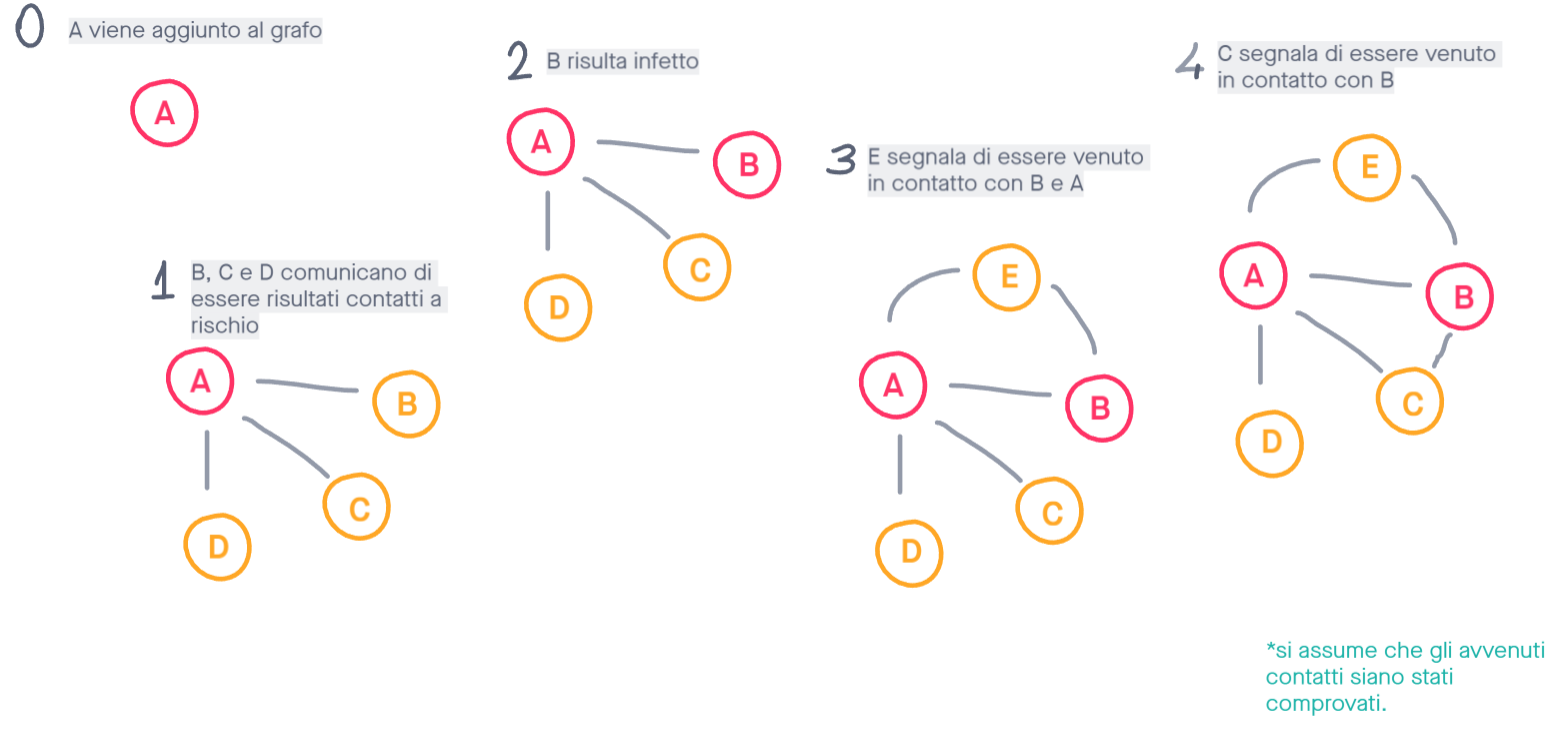


Figura 1: use case di costruzione del grafo di prossimità, che mostra l’evoluzione e l’aggiornamento del grafo mediante l’inserimento di archi tra un utente malato A con utenti a rischio B,C,D [step 1], l’update dello stato dell’utente B da uno stato di rischio a infetto [step 2] e l’inserimento di contatti tra utenti già mappati [step 3, step 4].

### ETICHETTE SUL GRAFO DI PROSSIMITÀ

Relativamente alla costruzione del grafo è possibile associare ad ogni arco delle etichette esplicative di informazioni relative al contatto tra i due utenti i cui nodi condividono quell’arco.

Etichette:

* *stima della durata del contatto*: il server può stimare la durata e il numero degli incontri tra A e B, contando quante volte B riesce a dar prova di un contatto con A e a quali epoche le prove fanno riferimento. In particolare, ogni H(ephIDi||i) è relativa a una fissata durata L. Se B comprova n connessioni, allora:
  + la durata totale dei loro incontri è dT = n × L;
  + la durata parziale di ogni incontro è dP = nP × L, dove np è il numero di epoche consecutive associate ai contatti comprovati da B;
  + il numero di incontri è nI = nT – nP, dove nT è il numero totale di epoche;
* *fattore di rischio*: gli smartphone, secondo DP-3T, quando trovano dei match con H(ephIDi||i) appartenenti a un cittadino infetto, computano localmente un fattore di rischio (i.e. stima della durata del contatto, distanza con l’utente infetto);
* *data del contatto*: data in cui A e B si sono incontratie lo smartphone di B ha salvato in locale H(ephIDi||i)||xi.

### Esempio

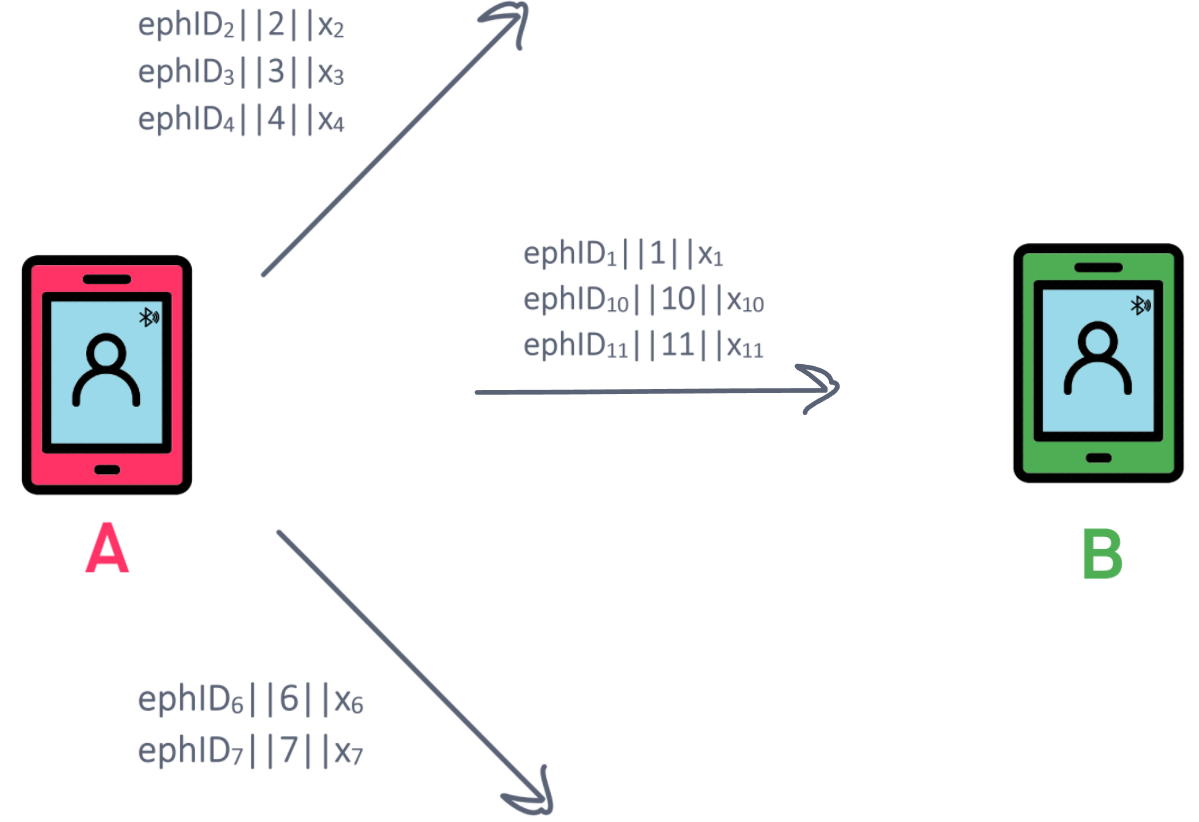


Figura 2: A manda in broadcast ephIDi||i per diverse epoche i. A e B si incontrano nelle epoche 1,10,11.

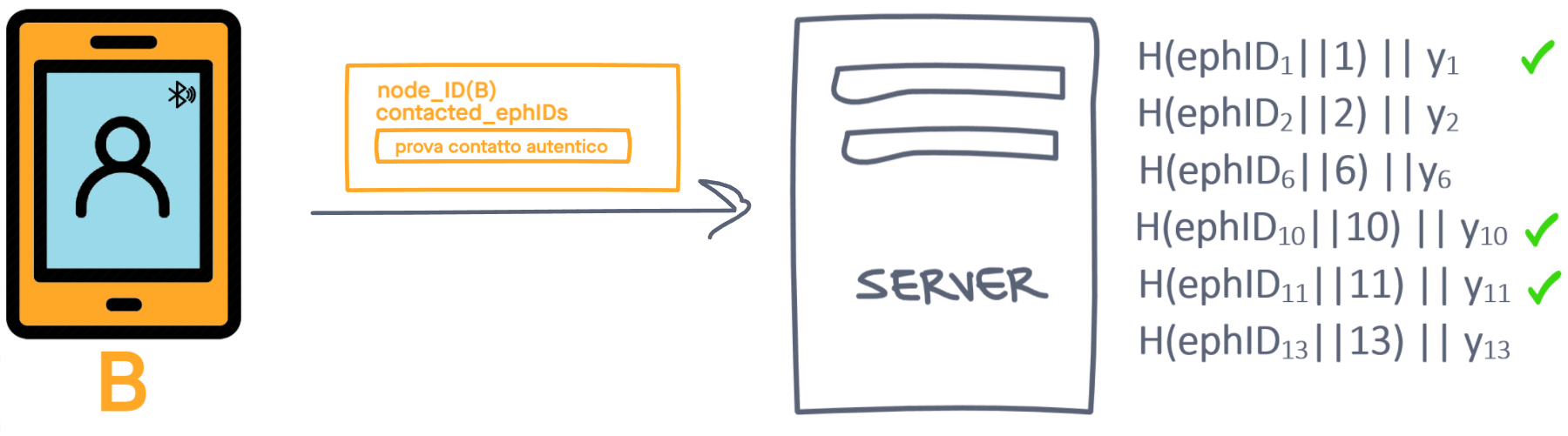


Figura 3: B trovando i match con H(ephIDi||i) per i = 1,10,11 viene identificato come utente a rischio. Dà quindi prova dei contatti al server.

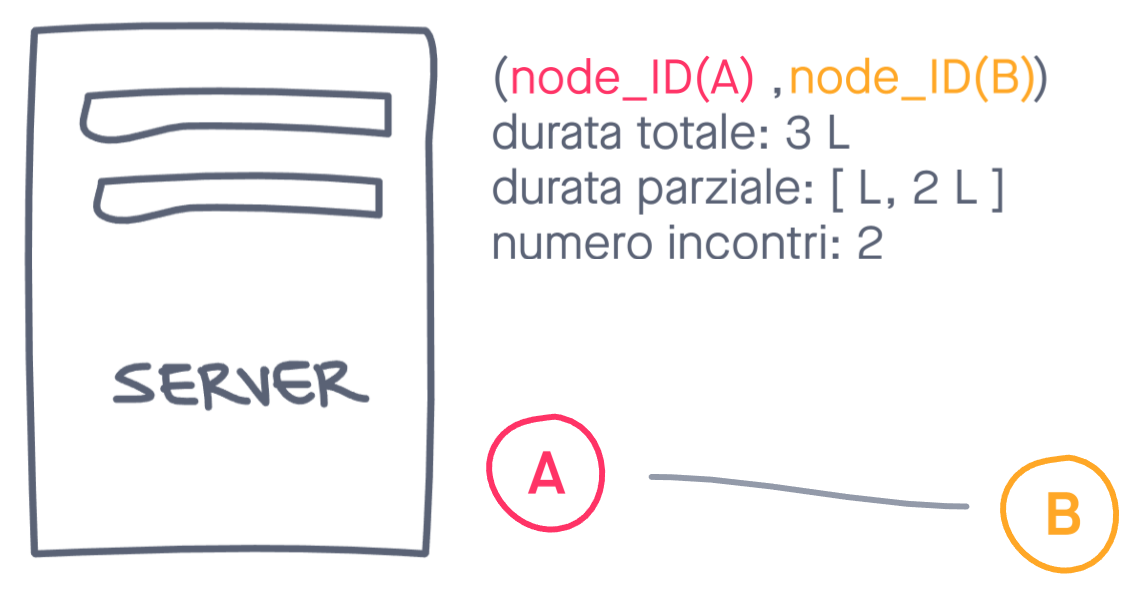


Figura 4: sulla base delle informazioni ricevute, il server calcola la durata totale degli incontri tra A e B, le durate parziali e il numero di incontri.

## 4. ANALISI DELLA CONFIDENZIALITÀ E DELLA INTEGRITÀ DEL SISTEMA PROPOSTO

### GOSSIP ATTACK

L’obiettivo dell’avversario è fare in modo che il sistema lo riconosca come utente a rischio in assenza di un reale contatto con un utente infetto. In particolare, l’avversario potrebbe voler corrompere l’integrità del grafo di prossimità oppure trarre benefici come:

* sospensione giustificata dal lavoro;
* maggiore priorità rispetto agli altri utenti nell’effettuare il test.

In mancanza di un protocollo che permetta al server di verificare che ci sia stato un reale contatto tra un utente infetto e gli altri utenti, un avversario potrebbe mentire al server comunicandogli di aver avuto un incontro con un malato e quindi di essere a rischio.

### MITIGAZIONE

Il protocollo di *zero knowledge proof* riesce a mitigare questo attacco esigendo una prova che attesti il reale incontro. In questo modo l’unica possibilità di falsificazione di un contatto è limitata alla cattiva condotta di un numero finito di utenti che potrebbero utilizzare il segreto ricevuto dal malato, precedentemente condiviso a un singolo utente, risultando così tutti a rischio. In ogni caso, nella pratica, la maggior parte degli utenti, beneficiando dei servizi offerti dall’applicazione, non tenta di sabotare il sistema limitando questo tipo di attacco a una ristretta cerchia di avversari.

### Esempio *gossip attack*

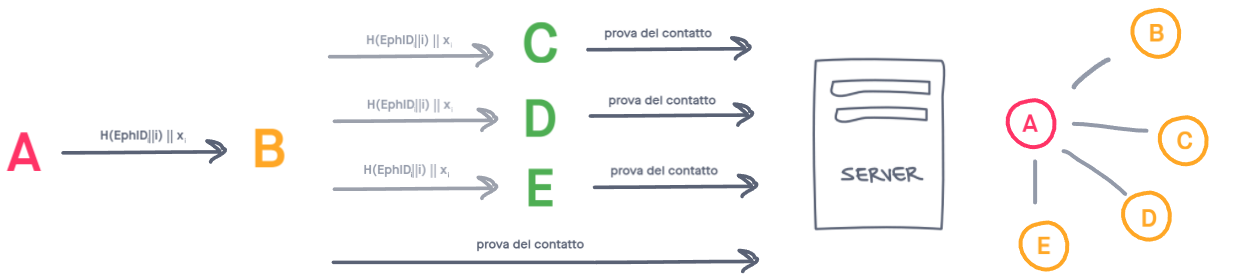
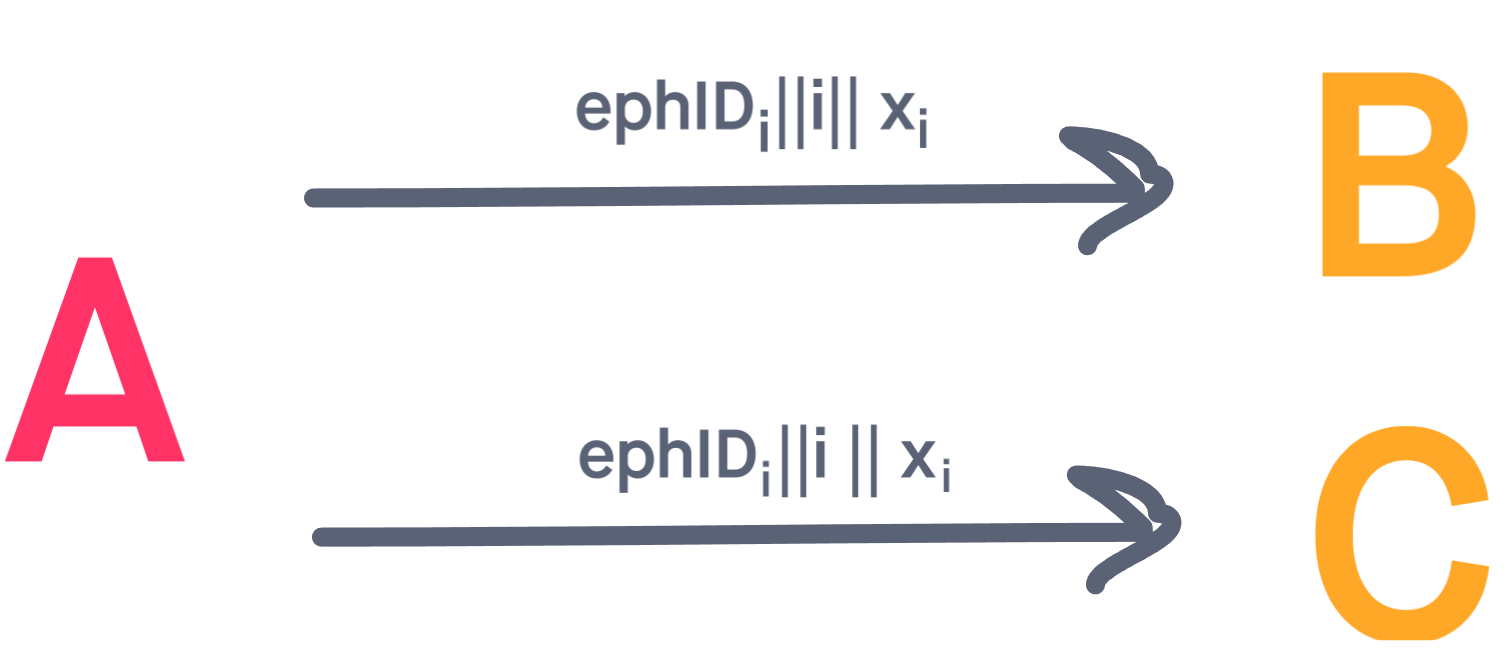


Figura 5: scenario in cui A è malato e ha un contatto con B che inoltra il messaggio ricevuto a C, D, E che ora possono comunicare al server di essere utenti a rischio.

### FAKE INFECTION SPREAD ATTACK

Sia (ephIDi||i)||xi il messaggio che l’utente malato A manda in broadcast e che sia B sia C ricevono.



Ci troviamo quindi nella situazione in cui è possibile da parte di un avversario sfruttare il fatto che xi è uguale sia per B sia per C, sia per tutti quegli altri utenti che ricevono nella stessa finestra temporale il medesimo messaggio di A. In particolare, il numero di utenti che ricevono lo stesso messaggio da A non è prevedibile. Pertanto, il server non è a conoscenza del numero massimo di utenti che possono essere realmente entrati in contatto con A e che quindi possono comprovarlo. Si distinguono dunque due possibili attacchi:

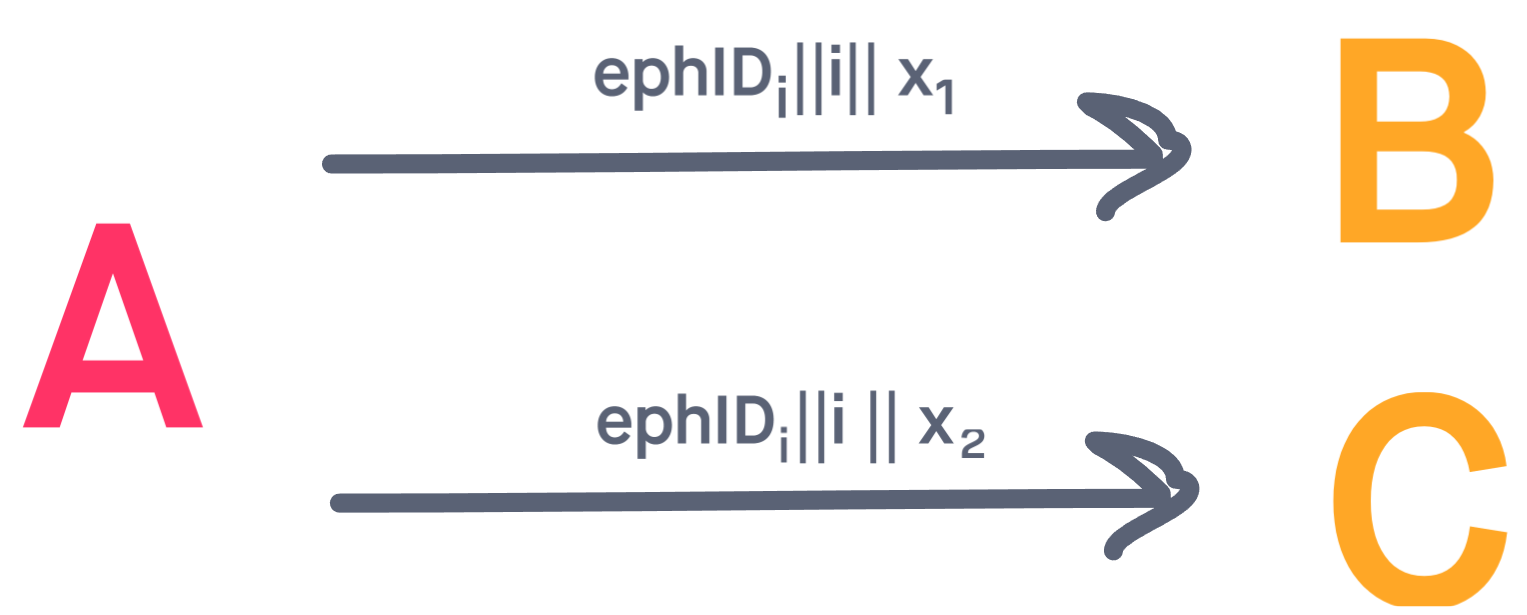
1. l’avversario è B o C o qualsiasi altro utente che ha ricevuto il messaggio, che può servirsi del protocollo di prova del contatto autentico per un numero indefinito di volte, cambiando ogni volta il campo *node\_id* nel pacchetto

*B->Server*.

1. B o C o qualsiasi altro utente che ha ricevuto il messaggio ephIDi||i||xi da A, può inoltrare il segreto xi ad altri utenti che possono a loro volta utilizzarlo indefinitamente per provare falsi contatti.

Dunque, sul grafo verrebbero mappati tanti nodi diversi, con corrispettivi *node\_id* diversi, quanti sono i pacchetti che il server riceve nonostante il contatto reale sia stato unico.

### MITIGAZIONE



Una soluzione ottima, ma non praticamente concretizzabile, previene questi tipi di attacchi differenziando il segreto che A scambia con B con quello che A scambia con C rispettivamente contrassegnati in figura con x1 e x2.

In questo scenario il server sa che per ogni contatto esiste ed è unico il segreto che A scambia con ogni altro singolo utente. Dunque, il server sfruttando questo meccanismo aggiungerà al grafo di prossimità solamente il nodo relativo all’utente che per primo comproverà il contatto. Inoltre, in seguito a tentativi multipli di dar prova per un medesimo contatto, il server potrebbe insospettirsi e seguire ulteriori protocolli di sicurezza (i.e. potrebbe salvarsi localmente i node\_id degli utenti che cercano di dar prova del contatto quando è già stato comprovato da un altro utente e tenerne conto).

Questa proposta tuttavia rimane teorica perché realizzarla comporterebbe un eccessivo consumo energetico che inficerebbe il normale utilizzo degli smartphone scoraggiando gli utenti a utilizzare l’applicazione.

### RELAY ATTACK

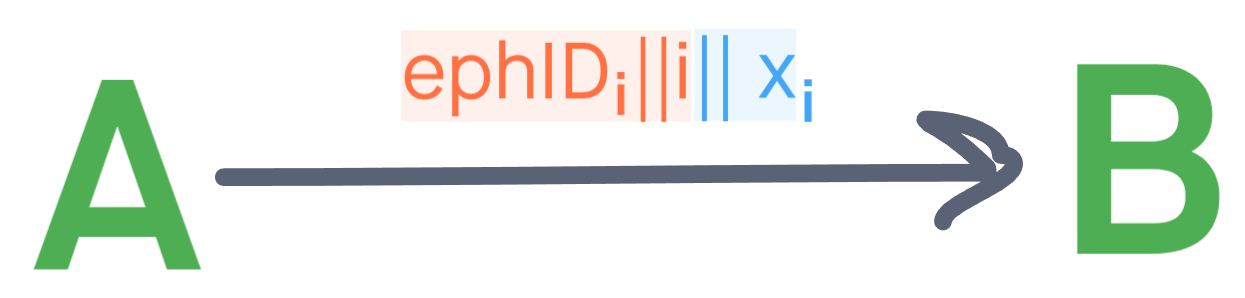
L’obiettivo dell’avversario è di compromettere l’integrità del grafo di prossimità facendo in modo che una parte dell’utenza riceva notifiche fake di contatti con utenti infetti non realmente avvenuti. In particolare, ciò è possibile posizionando antenne in postazioni pubbliche (i.e. aeroporti, stazioni dei treni, centri commerciali) capaci di trasmettere il segnale lontano e ad ampio raggio.

### MITIGAZIONE

Come riportato nel paper *“Hashomer – A Proposal for a Privacy-Preserving Bluetooth Based Contact Tracing Scheme for Hamagen Benny Pinkas∗ Eyal Ronen† Version 0.93 – April 27, 2020”* una possibile soluzione sta nell’incorporare nel messaggio BLE la cifratura del GPS relativa alla posizione in cui è avvenuto il contatto e firmarla con un protocollo MAC.

### TRADE OFF |ephIDi| |xi|

Il protocollo DP-3T prevede che per codificare ephIDi||i vengano impiegati 16 byte.



Affinché si rispettino i limiti dimensionali imposti dalla connessione BLE e sia possibile concatenare all’ephIDi il segreto xi, i byte vengono così suddivisi:

|  |  |
| --- | --- |
|  | |ephIDi||i| = 11 byte  |xi|= 5 byte |

Nella scelta di come suddividere i 16 byte tra i due campi si è cercato di trovare un trade-off tra |ephIDi||i | e |x|:

1. quando |ephIDi||i| viene ridotto, necessariamente Pr[ephIDj = ephIDk] aumenta, per qualsiasi j≠k;
2. quando |xi| viene ridotto, la facilità con cui un avversario può fare brute force su xi aumenta.

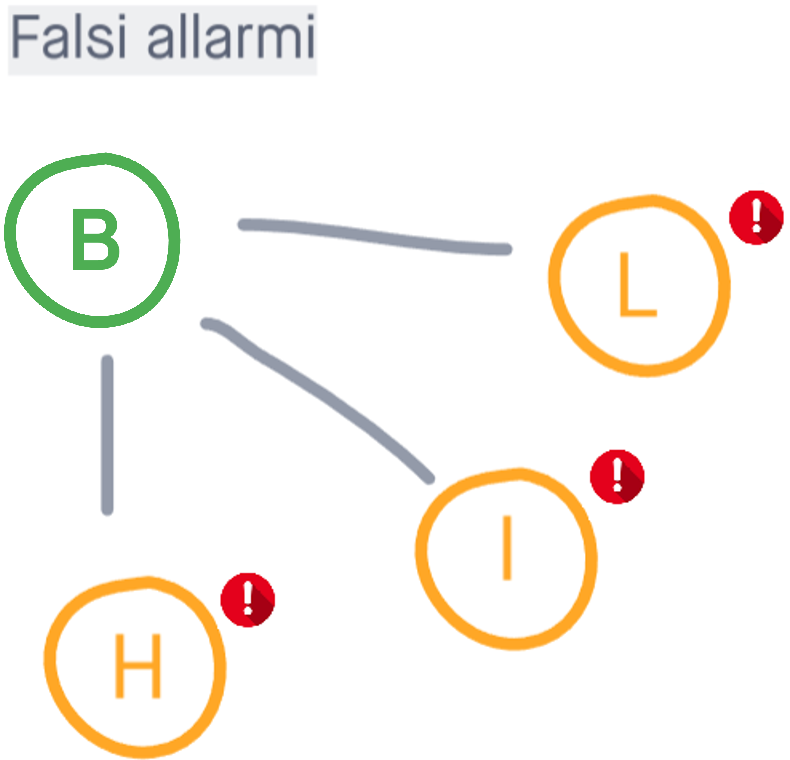
### Esempio scenario 1

Sia ephIDi||i||x1 il messaggio inviato in broadcast da A e ricevuto da E, F, G, e sia ephIDj||j||x2 il messaggio inviato in broadcast da B e ricevuto da H, I, L.

|  |  |
| --- | --- |
|  |  |

E sia inoltre ɛ una negligible function e Pr[ephIDj||i = ephIDk||i)] ≥ ɛ ∀ j≠k allora, qualora A fosse malato, gli utenti a rischio risulterebbero E,F,G,H,I,L.

In particolare, H,I,L in questo caso troverebbero un match tra le H(ephIDj||i) inviategli dal server e le H(ephIDk||i) che avevano salvato in locale, deducendo erroneamente di esser utenti a rischio.



Tuttavia, questo non inficerebbe la corretta costruzione del grafo di prossimità poiché H,I,L non riuscirebbero a comprovare un contatto con A, avendo ricevuto da B come segreto x2≠x1.

Esempio scenario 2 (aggiungi 2^potenza)

Qualora |xi| sia sufficientemente piccolo da permettere a un avversario un attacco di brute force, questi, in assenza di contromisure da parte del server, potrebbe tentare tutti i possibili valori di xi e riuscire a comprovare un contatto mai avvenuto.

Sulla base delle precedenti considerazioni, risulta vantaggioso ridurre |ephIDi||i| il minimo necessario da non consentire un attacco di brute force su xi.

### CONSIDERAZIONI SUI VANTAGGI DELLA ZERO KNOWLEDGE PROOF

Il sistema proposto eredita da DP-3T l’idea di un meccanismo secondo il quale un cittadino infetto viene certificato come tale da un’autorità sanitaria, la quale provvede a fornire al paziente un codice di autorizzazione inizialmente disabilitato che diventa attivo nel momento in cui il suo test al COVID risulta positivo. In particolare, l’utente infetto può utilizzare il codice fornitogli per comunicare al server il suo stato di salute partecipando volontariamente alla costruzione del grado di prossimità.

Da quanto evinto dal paper di DP-3T è legittimo dedurre che esista un’associazione tra l’identità del cittadino e il codice di autorizzazione fornitogli dall’autorità sanitaria. Anche se il server non ricostruisse mai la corrispondenza identità-codice, l’autorità sanitaria ne sarebbe comunque a conoscenza.

Un’istituzione giuridica potrebbe essere autorizzata per legge a richiedere le coppie (identità, codice) all’autorità sanitaria e ad accedere al grafo di prossimità gestito dal server. Non escludendo la possibilità che il server salvi i codici di autorizzazione fornitigli, l’istituzione giuridica, oltre a conoscere l’identità dei malati, avrebbe prova, in assenza del protocollo di *zero knowledge proof*, dei contatti tra gli utenti infetti.

### Esempio

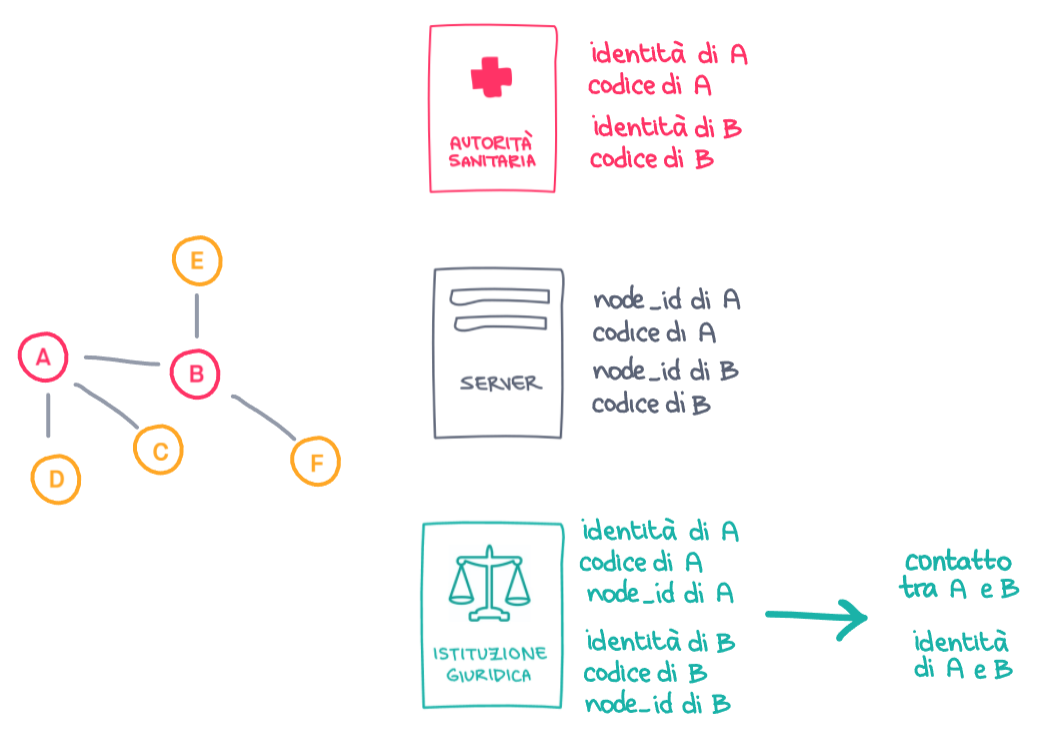


Figura 6:scenario in cui, in assenza di proof of knowledge, un’istituzione giuridica ricostruisce un contatto tra A e B, utenti malati, di cui conosce l’identità.

### MITIGAZIONE

Grazie al protocollo di *zero knowledge proof*, l’autorità giuridica non può servirsi del grafo di prossimità per attestare un contatto tra A e B.

### VULNEARBILITÀ SUL CODICE DI AUTORIZZAZIONE

Si assume che il codice di autorizzazione che un utente malato fornisce al server sia spendibile una sola volta e consegnato a un cittadino infetto esclusivamente da un’autorità sanitaria.

### VULNERABILITÀ SULL’INTEGRITÀ DELLE ETICHETTE SUL GRAFO DI PROSSIMITÀ

In riferimento alle etichette associate agli archi sul grafo di prossimità distinguiamo diversi livelli di affidabilità. In particolare, assumendo che il protocollo della *prova del contatto* sia sicuro, allora la *stima della durata del contatto* computata dal server sarebbe affidabile. In contrapposizione, l’integrità delle informazioni aggiuntive che l’utente potrebbe caricare sul server (i.e. *data di contatto, fattore di rischio*) dipende esclusivamente dall’utente che le comunica spontaneamente al server.

## IMPLEMENTAZIONE

L’obiettivo è stato implementare il protocollo di *zero knowledge proof* secondo il modello di *Schnoor* non interattivo. In particolare, viene utilizzato il gruppo *1024-bit MODP Group with 160-bit Prime Order Subgroup* come descritto nella sezione dedicata alla *zero knowledge proof*.

### DIRECTORT TREE

|  |  |
| --- | --- |
|  | DIRECTORIES:  * *apache2\_conf\_files;* * *certificates;* * *client\_scripts;* * *env;* * *zero\_proof\_knowledge;* * *zkp\_server/zkp\_server;* |

#### *APACHE2\_CONF\_FILES*

Directory in cui si trovano i file di configurazione di apache:

* *apache2.conf*
* *000-default.conf*
* *default-ssl.conf*

#### *CERTIFICATES*

Directory in cui si trovano:

* *servercert.pem*: certificato del server;
* *serverkey.pem*: chiave usata nel certificato del server;
* *cacert.pem:* certificato della CA che ha rilasciato al server server.pem;
* *reqserver.pem:* richiesta di certificato del server;
* *openssl.cnf:* file di configurazione per la generazione dei certificati.

#### *CLIENT\_SCRIPTS*

Directory in cui si trovano:

* *client.py*: script python che simula un client che, con una richiesta POST, instaura, su connessione https, il protocollo di *zero knowledge proof* col server;
* *group\_utilities.py*: script python tramite cui il client istanzia un oggetto della classe *Prover* con cui instaurare il protocollo di *zero knowledge proof* con un oggetto della classe *Verifier* istanziato dal server. Compito del *Prover* è inviare al *Verifier* la prova di conoscenza del segreto.

#### *ENV*

Directory del virtual environment in cui sono stati installati i package utilizzati nei moduli costituenti l’implementazione del progetto.

*ZERO\_PROOF\_KNOWLEDGE:*

applicazione django contenente:

*group\_utilities.py:* script python tramite cui il server istanzia un oggetto della classe *Verifier* con cui un oggetto della classe Prover, istanziato dal client, instaura il protocollo di *zero knowledge proof*. Compito del *Verifier* è verificare la prova di conoscenza del segreto inviatagli dal *Prover*.

*views.py*: script python in cui l’oggetto della classe *Verifier* istanziato dal server verifica la prova di conoscenza del segreto inviatagli dal *Prover*.

### ULTERIORI DETTAGLI

Gli script python si appoggiano a Django framework. Per l’interfacciamento al web server, realizzato con Apache, è stato utilizzato WSGI.

### CONNESSIONE HTTPS

Immagine che contiene screenshot

Descrizione generata automaticamente

Il protocollo di zero proof of knowledge si appoggia alla connessione https. In particolare, è stato inserito, tra i certificati fidati del browser, il certificato *cacert.pem* della CA che ha rilasciato al server il certificato *servercert.pem* .

### ESEMPI DI ESECUZIONE

#### SIMULATED = TRUE

Immagine che contiene screenshot, monitor, schermo

Descrizione generata automaticamente

Come si legge da riga 22, il parametro *simulated* è impostato a *True*. Ciò significa che indipendentemente dal parametro *secret*, il *Prover* genererà a run-time un segreto *x* e computerà l’informazione pubblica *y* che comunicherà al *Verifier*. L’esecuzione termina con successo. Il *Prover* conosce il segreto.

#### SIMULATED = FALSE

Immagine che contiene screenshot, monitor, schermo

Descrizione generata automaticamente

Come si legge da riga 22, il parametro *simulated* è impostato a *False* e al parametro *secret* è stato assegnato *x\_test*. A riga 30 si vede assegnato alla variabile *y* il valore contenuto in *y\_test*. Ciò significa che il *Prover* tenterà di dar prova di conoscere il segreto x\_test associato all’informazione pubblica *y\_test.* In particolare il valore di *x\_test* e y\_test coincidono rispettivamente con quelli di xA e yA riportati nella sezione ZERO KNOWLEDGE PROOF (G,p,g,q,x,y) relativamente al gruppo *1024-bit MODP Group with 160-bit Prime Order Subgroup.* L’esecuzione termina con successo. Il *Prover* conosce il segreto.

#### ZERO KNOWLEDGE PROOF FAILED

Immagine che contiene screenshot, monitor, schermo, portatile

Descrizione generata automaticamente

Come si legge da riga 22, il parametro *simulated* è impostato a *False* e al parametro *secret* è stato assegnato *x\_test*. A riga 30 si vede assegnato alla variabile *y* il valore contenuto in *y\_test*. Ciò significa che il *Prover* tenterà di dar prova di conoscere il segreto x\_test associato all’informazione pubblica *y\_test.* In particolare il valore di *x\_test* e y\_test coincidono rispettivamente con quelli di xA e yA riportati nella sezione ZERO KNOWLEDGE PROOF (G,p,g,q,x,y) relativamente al gruppo *1024-bit MODP Group with 160-bit Prime Order Subgroup.* L’esecuzione termina con successo. Il *Prover* conosce il segreto.