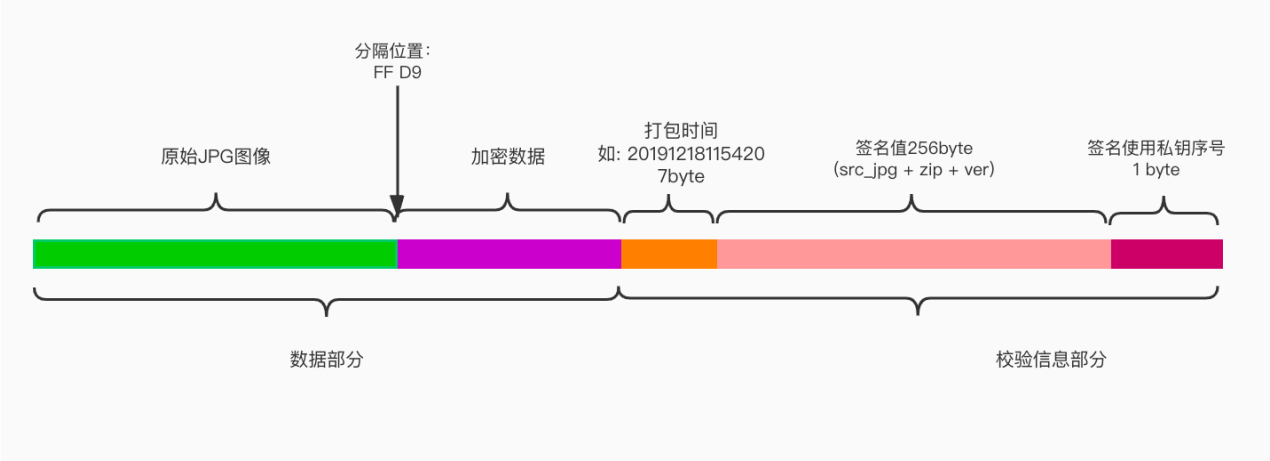
整体流程

1. 服务器端通过图片隐写形式将配置数据按照指定格式写入到图片中，并使用指定路径推送到各指定图床平台，具体配置数据见下方示例。
2. 客户端内置一张初始图片，根据配置项`interval`周期性拉取图片，并按照策略更新本地配置。
3. 当客户端发现配置中的某个proxy ip或者图片URL无法访问次数到达对应阈值后，通过firebase渠道进行数据上报
4. 运营侧根据firebase收集到的上报信息，根据实际决策结果更新指定保活图配置并执行新图推送

保活图结构示意



保活配置示意

配置中IP区域缩写请详见 https://countrycode.org/

```
{
  "interval": 300, // 图片拉取周期，单位秒
  "pr_th": 10,     // proxy ip访问错误多少次后上报日志
  "img_th": 10,    // 图片访问错多少次后上报日志
  "pub_imgs": [    //公网图片访问地址
    "https://account.github.io/dst/action-icon-white.jpg",
    "https://account.github.io/dst/icon-marketplace.jpg"
  ],
  "ip": [          //按下发地区分组的proxyip列表
    "COMMON":          // 下发给COMMON分组的IP列表（默认可用）
    [
      "1.1.1.1",
      "2.2.2.2"
    ],
    "IR":              // 下发给IR分组的IP列表（伊朗）
    [
      "3.3.3.3",
      "4.4.4.4"
    ]
```

```
  ]
}
```

制图流程示例

1. 根据实际配置信息按照以上示例格式生产json
2. 随机挑选一组aes key/iv，对json进行加密到文件 list.data，然后将key/iv对应的序号输出到文件 enc.txt
3. 将list.data和enc.txt打包到一个zip文件中，如 targe.zip
4. 将zip文件追加至目标文件如src.jpg末尾，生成图片target.jpg
5. 使用当前系统时间作为当前的"版本号"，并追加至target.jpg末尾
6. 随意挑选一个rsa密钥，并对target.jpg进行签名计算，然后将签名值、rsa密钥索引号依次追加到target.jpg末尾

bash制图示例

```
#list.json加密,aes256-cbc
openssl aes-256-cbc-e -K
E30129D59ABB86242854E54C673244E235730723DDD9DF774E2F70EB065BD20B -iv
81497271F209BD29056E32675ACEC4C9 -in list.json -out \$

#记录使用的密钥编号到指定文件
echo "1" > \_

# 打包加密数据
zip target.zip \$ \_

# 向图片中追加压缩文件信息并输出target.jpg
cat src.jpg target.zip > target_1.jpg

# 使用当前时间作为 版本号 数据
date "+%Y%m%d%H%I%S" | xxd -r -p >> target_1.jpg

# 签名计算
## 挑选一个rsa密钥并记录索引序号
echo 1 | xxd -r -p > rsa_idx
## 签名
dgst -sign key1.pub -sha256 -out sign.txt target_1.jpg

# 组装最终图片
cat target_1.jpg sign.txt rsa_idx > target.jpg
```

golang验证示例

```
func main() {
  data, err := ioutil.ReadFile(os.Args[1])
  if err!=nil {
    fmt.Printf("文件打开失败: %v\n", err)
    return
  }
  n := new(big.Int)
  fileLen := len(data)
  rsaIdxSrc := n.SetBytes(data[fileLen-1:fileLen])
  rsaIdx := int(rsaIdxSrc.Int64())
  if _, ok := rsaKeys[rsaIdx]; !ok {
    fmt.Printf("rsa公钥索引不存在: %d\n", rsaIdx)
    return
  }

  rsaSign := data[fileLen-257:fileLen-1]
  verStr := data[fileLen-264:fileLen-257]

  signData := data[0:fileLen-264]
  signData = append(signData, verStr...)

  publicKey := []byte(rsaKeys[rsaIdx].PublicKey)
  if !rsaVerySignWithSha256(signData, rsaSign, publicKey) {
    fmt.Println("rsa签名验证不通过")
  } else {
    fmt.Println("RSA签名验证通过")
  }

  fmt.Printf("打包时间: %x\n", verStr)

  var zipPos = 0
  for ; zipPos<fileLen-284; zipPos++ {
    if fmt.Sprintf("%x", data[zipPos:zipPos+2]) == "ffd9" {
      break
    }
  }
  if zipPos >= fileLen-284 {
    fmt.Println("无法截取出zip文件")
    return
  }
  fmt.Printf("zipPos: %d\n", zipPos)
  zipFile := data[zipPos+2:fileLen-264] // 后284位为校验部分固定长度
  files, err := unzipData(zipFile)
  if err != nil {
    fmt.Println("解压zip数据失败")
    return
```

```
  }
  checkFiles := []string{`$`, `_`}
  for _, f := range checkFiles {
    if _, ok := files[f]; !ok {
      fmt.Printf("压缩包缺少关键文件: %s\n", f)
      return
    }
  }
  aesIdxStr := string(files[`_`])
  if _, ok := aesKeys[aesIdxStr]; !ok {
    fmt.Printf("aes key索引不存在: %s\n", aesIdxStr)
  }
  key, _ := hex.DecodeString(string(aesKeys[aesIdxStr].Key))
  iv, _ := hex.DecodeString(string(aesKeys[aesIdxStr].Iv))
  content, err := aesDec(files[`$`], key, iv)
  if err != nil {
    fmt.Printf("解密失败: %v\n", err)
    return
  }
  fmt.Println("===============[ 以下是json内容 ]===============")
  fmt.Printf("%s\n", string(content))
  return
}
```

AES

序号	key	iv
1	B9BE056BDC6AD2994536524B3EC2CB69696EAB6A1EE4723DC5E57709345667E2	F2AF781CB00EA2D146F088159DF4E5D4
2	2FF7BF36D7AB2790C0744362062DEF71B3905E8F3F86277D1504DC2F010F9C2E	02475958B8EE6D06491F975960118656
3	2F63EF49365C72A3BC56405C329FB6D1DF52513680ED1127CD88A6507E68ECBC	03F723FE982227374360D4A8C7C728D2
4	57C1B4E73CB8E668158527FFF43E41BD69BE454686E1D4FA84F92EBD2946DB19	8ABA5FC8DA067C06D7CB44F09FD654BA
5	FD44F8967DB5C6D95BF36049E20488FA96F147B6768A801DD80BDD5732A133A7	4B745C85A5D55C7060A1D39AFA101E8B
6	C5D5187485238765DAA4F8BBD2F69D2097958E1407EFCDFB9769F30E0C49EABA	2265C7A24C51E80311A51D961D24AA63
7	BD574AD5DAFEF7411AD7A25B16B365D43891FA84B895F6D65B987F14EA2163BA	0DC7D16ED3E8CF1A266E030CDB7422A8
8	6DBCB9904EF481F0B5FA90201D4DDD3CE59E311471487DDF5C12ACB1E983E420	5E41DF9DE2E4579F946B035FCB1B4AF3
9	629C495CBE0BC7176ED1966845254CED3E6EA7FAC30F7418A287AD9CB21C7901	7ED98CE5AF07EF629BE5F738C2D9643D
10	5E6C9C0859A986D9DA2A503AB38E137EBF7B0E30A83EE1527500F5837BFEEDE8	BB32110BC6B26D2B432DC07168418ECB

RSA

序号	公钥
1	-----BEGIN PUBLIC KEY----- MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu7qw+GucKLOXIA/iPcKE t5tQtJis0aINHKfi6Sf1zlyZp1oL9U1l0GL3uAlGeOL4imm4NeebKzHAjPHcYyTI 9yOHMC9oFtNEG1UCna9on82gRLIRwpgYIJE/h/3ZPVBV6nMM0G3ZeuIFDCHlQ685 9uMAWHrRicMgZ+DZePsHI9ZYkYk2rezo/67T/zZqwVGlHPYc2qrVof0wITQ5IJvB

kB/0UBKPpiYbaf9aH6FmLIzaUTy7kHXLcjANR+H2JI+xeZvN0sMUF3z5aAyjH+he
cTo9tTwzGsa+p+TTmrnR/EWeWNUF2DykvpKeLtkIXYUCA54Z23NQTyoMVpJz71/l
XQIDAQAB
-----END PUBLIC KEY----- |
| 2 | -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu9NbB01YompF1ogLwn/T
LFY5OvFjouWiedex9IOl4jTqF/uiwHZMDQWkD9VxjmYNSdXtwuHdNWGD0W70vrZX
0kI7tORMQ+OXze1CSdwS1diytV9zOgAC83d3mORrGNC6wLOqkaHFNTqX2Mpc/Wlo
s/OfewlhqMR8yM8W/pNLfUCTjI8exGRIlQO4rzIVNFaT3ccwFS2FkXC9Uk+8Xwnz
thxb1ENzOGcqQndErvaKeBM7DkYsbEW87qUTQepZFGMFLJ1OiqMcYnyYQv2GuQQI
ZsOsyRaveBJBXSmEKxzDwPYDYIiX+LncUOgdjIifFsJkXG/xyf0CnyfkdKa+Y2EA
EQIDAQAB
-----END PUBLIC KEY----- |